Escuela Politécnica Superior

Twilight Wispers: Videojuego en Unity3D

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Alberto Salieto Gómez

Tutor:
Miguel Ángel Lozano Ortega

Universitat d'Alacant
Universidad de Alicante

Septiembre 2017

DEDICATORIA

Gracias a los profesores que se han volcado en la enseñanza a lo largo de la carrera, a aquellos que no sólo han enseñado, sino que han sabido despertar las ganas de seguir aprendiendo.

Especialmente a mi tutor Miguel Ángel por haberme guiado y aconsejado en todos los ámbitos.

Gracias a mi familia por haberme apoyado incondicionalmente, y a los compañeros de carrera que ahora puedo considerar amigos, por haber compartido tantas experiencias en esta etapa, especialmente a Albert, David y Alex.

“Si siempre lo has hecho de esa manera, probablemente esté mal.”

- *Charles Kettering*

RESUMEN

Este proyecto consiste en desarrollar un videojuego del género estrategia y tower defense desde cero para dispositivos de sobremesa, que disponga de distintas mecánicas que presenten retos a superar. Para ello se utilizará el motor gráfico Unity3D, uno de los más utilizados a la hora del desarrollo de videojuegos, debido a su gran potencial. El jugador controlará a una hechicera que deberá impedir la llegada de seres oscuros a su reino, defendiéndolo por medio de hechizos y estructuras defensivas.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Hoy en día el mundo de los videojuegos se ha adaptado firmemente entre nosotros, siendo ya algo común en el día a día de nuestra sociedad. Gracias a la evolución del hardware y de los recursos disponibles, los videojuegos ya no sólo son una simple herramienta de entretenimiento, por lo que no es de extrañar que encontremos videojuegos de múltiples ámbitos dirigidos a entretener, enseñar, divertir, y por supuesto, transmitir.

Podemos decir, por tanto, que tratan de sorprendernos innovando y evolucionando cada día para seguir mejorando en todos sus aspectos: con nuevas e innovadoras mecánicas de juego, mejorando aspectos gráficos, cinemáticas, efectos especiales realistas, historias que son capaces de conectar con los jugadores o contando con bandas sonoras equiparables a las utilizadas en el mundo del cine. El resultado de todos estos componentes esenciales de un videojuego puede resultar en desde sacar a millones de personas a caminar por su ciudad en busca de algún Pokémon hasta dejar huella en el jugador con historias que consiguen emocionar, o sencillamente entretenernos.

Además, la evolución de la tecnología y las facilidades y recursos disponibles ofrecidas por las distintas plataformas y compañías del sector hacen que los pequeños desarrolladores y diseñadores o equipos de desarrollo puedan trabajar en ellos de forma multidisciplinaria, con tareas de diseño, desarrollo y arte.

Este documento contiene lo relativo al proyecto **Twilight Whispers**, basado en la creación de un videojuego 3D para dispositivos de sobremesa en utilizando el motor de juego Unity3D. El juego es del género estrategia-tower defense, basado en el uso de estructuras defensivas en el nivel. Este documento, pues, presenta la motivación que lleva a la elección de este tema, su diseño, implementación, organización del proyecto y resultado final.

Finalmente, en forma de anexo, aparecen referencias a los recursos utilizados y de capturas del resultado final del proyecto in-game.

2. MARCO TEÓRICO O ESTADO DEL ARTE

En este bloque analizaremos brevemente el concepto básico de un videojuego, sus orígenes y evolución hasta su estado actual en la industria moderna del ocio y entretenimiento, así como algunos ejemplos que han motivado este proyecto y que han servido de referencia en diferentes sentidos.

2.1. CONCEPTO DE VIDEOJUEGO

Antes de entrar en materia acerca del proyecto, nos aproximaremos a él comprendiendo su definición más simple. La definición básica que encontramos en fuentes de información como Wikipedia o la RAE encontramos lo siguiente:

"Un videojuego o juego de video es un juego electrónico en el que una o más personas interactúan, por medio de un controlador, con un dispositivo que muestra imágenes de vídeo."

Sin embargo, este término queda pequeño si acudimos a todo lo que engloba, como medio de entretenimiento y de comunicación. Los videojuegos llevan instaurados en la sociedad desde hace décadas, pero la aparición de los juegos independientes, el mercado creció exponencialmente. Al no depender de un gran equipo ni una gran inversión para poder crear un videojuego, aparecieron cientos de herramientas, plataformas y dispositivos que se adecuaban a este nuevo mercado. Plataformas como Steam, Origin, y herramientas de desarrollo y soporte, como Unity3D, Gamemaker o Unreal Engine potenciaron el llamado "boom de los videojuegos."

Este "boom" también popularizó mecánicas poco conocidas, sacando a relucir algunas simples en apariencia que en desarrollos pasados se reducían al mínimo para adecuarse a las capacidades del hardware de la época. Reducir las mecánicas de un juego a sencillamente saltar y disparar a veces era necesario para adaptarse a los requisitos disponibles.

Con el paso del tiempo, se ha conseguido enfocar cada videojuego en diferentes ámbitos, resaltando cualidades específicas de cada uno, ya sea centrándose en desarrollar mecánicas innovadoras, añadiendo mejoras gráficas y visuales o creando narraciones que consigan cautivar al jugador, como bien podría hacerlo una película o un libro, pero inmergiendo al jugador dentro del universo creado por el videojuego.

2.2. MOTORES

Debido a la expansión en el ámbito de los videojuegos, existen diversas plataformas en las que podemos desarrollar un juego. Actualmente, las herramientas más populares para el desarrollo son motores de videojuegos que permiten la creación prácticamente completa de un proyecto.

Hay multitud de herramientas disponibles, pero en este documento haremos un breve resumen de los 2 motores más usados hoy en día y sus características principales.

2.2.1. UNITY 5

Figura 1. Logo de Unity

Unity es un motor de videojuegos multiplataforma que incluye diversos motores que facilitan el desarrollo e implementación de un proyecto. Contiene motor de renderizado, un motor físico, scripting, audio, importación y compilado y diversas características que lo hacen muy atractivo para implementar un proyecto tanto en equipo como en solitario, ya que facilita cada apartado de forma sencilla. Es ideal para la creación de prototipos, basado en componentes y acepta tanto C# como Javascript.

Sus principales características incluye un sistema de integración multiplataforma que permite exportar juegos para multitud de dispositivos y plataformas, siendo así uno de los motores más empleados para desarrollo de juegos de plataforma móvil, pero con la capacidad de desarrollo para Playstation, Xbox, WiiU y navegadores web.

Además, dado su extenso uso cuenta con una extensa documentación y soporte tanto por miembros del motor como por miembros de la comunidad, por lo que el soporte está prácticamente garantizado.

En cuanto a licencias, Unity cuenta con diferentes versiones, contando con una gratuita (personal) y otras suscripciones de pago como la profesional. La principal limitación de la versión personal es

una corta pantalla de carga con el logotipo al comienzo de la aplicación y que se limita el desarrollo para consolas y otros servicios adicionales como Unity Cloud Build.

Con respecto a las desventajas, no está muy orientado a ser usado por diseñadores o modeladores, y su versatilidad para el desarrollo en diferentes plataformas es activada con licencias premium (de pago), que pueden ser una barrera en el desarrollo.

Figura 2. Interfaz de Unity3D.

Debido a la versatilidad de Unreal y el conocimiento previo del mismo, será la herramienta principal de desarrollo, ya que permite juntar todos los recursos que componen el proyecto, utilizando la versión gratuita.

2.2.2. UNREAL ENGINE 4

Figura 3. Logo de Unreal Engine.

Unreal Engine 4, originalmente desarrollado principalmente como un motor de videojuegos orientado para los FPS (First Person Shooter), es un motor de videojuegos que utiliza C++, que se ha popularizado mucho en los últimos años gracias a sus extraordinarias capacidades gráficas y a su sistema de Blueprints, un sistema que puede usarse para programar sin escribir código gracias a su interfaz. Estos blueprints pueden exportarse, lo que potencia los cambios y colaboraciones entre desarrolladores.

Entre sus capacidades gráficas, incluye avanzados sistemas de iluminación dinámica y un complejo sistema de partículas capaz de manejar ingentes cantidades de partículas en una misma escena.

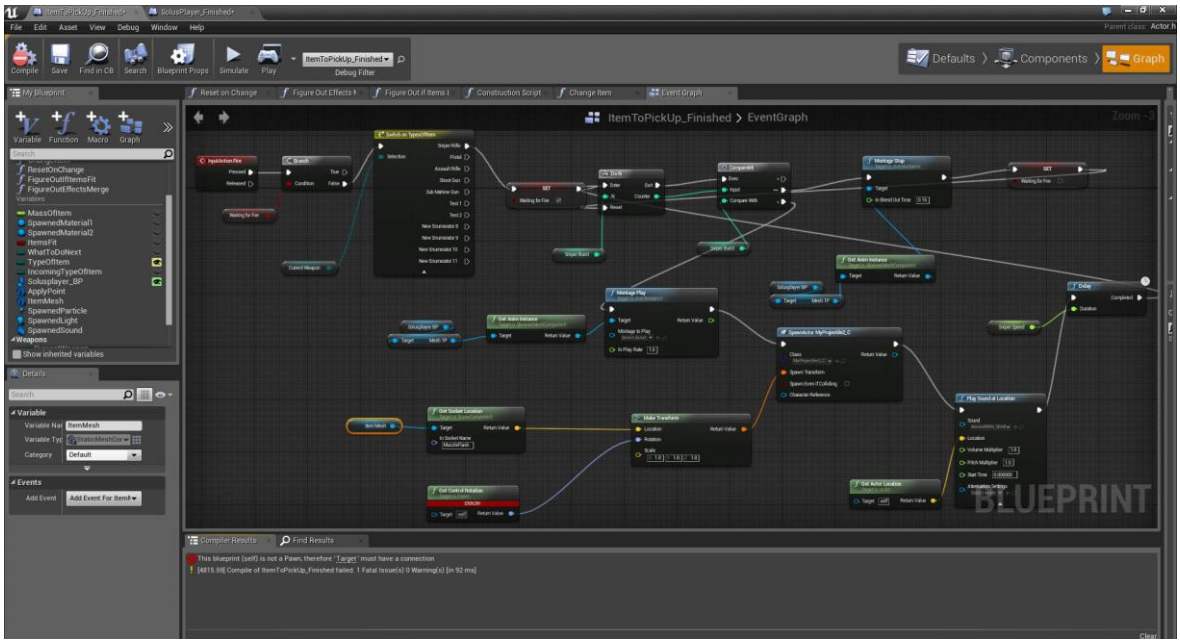


Figura 4. Blueprints en la interfaz de Unreal Engine 4.

En cuanto a sus desventajas, está principalmente orientado a equipos grandes de calidad, generalmente videojuegos o proyectos de sobremesa debido a su potencia, por lo que queda demasiado grande para algunos proyectos pequeños o dirigidos a plataformas móviles o portátiles.

2.3. ESENCIA Y REFERENCIAS

Como mencionamos, el aumento de recursos disponibles por los desarrolladores y el auge en el sector han impulsado los diferentes géneros que encontramos hoy en día en el mundo de los videojuegos. Ya no sólo en géneros (shooters, plataformas, aventuras gráficas), sino que cada uno trata de potenciar ciertos aspectos que destaquen dentro del propio desarrollo, la "esencia" de cada juego.

Destacaré algunos juegos generalmente conocidos, que brillan por tener un contenido muy simple pero destacable que es suficiente como para abarcar todo el sistema de juego en sí.

2.3.1. PORTAL

El primer ejemplo de apartado simple pero innovador lo encontramos en Portal, donde nos encontramos en unos escenarios realmente simples con una mecánica innovadora: crear portales de teletransporte.

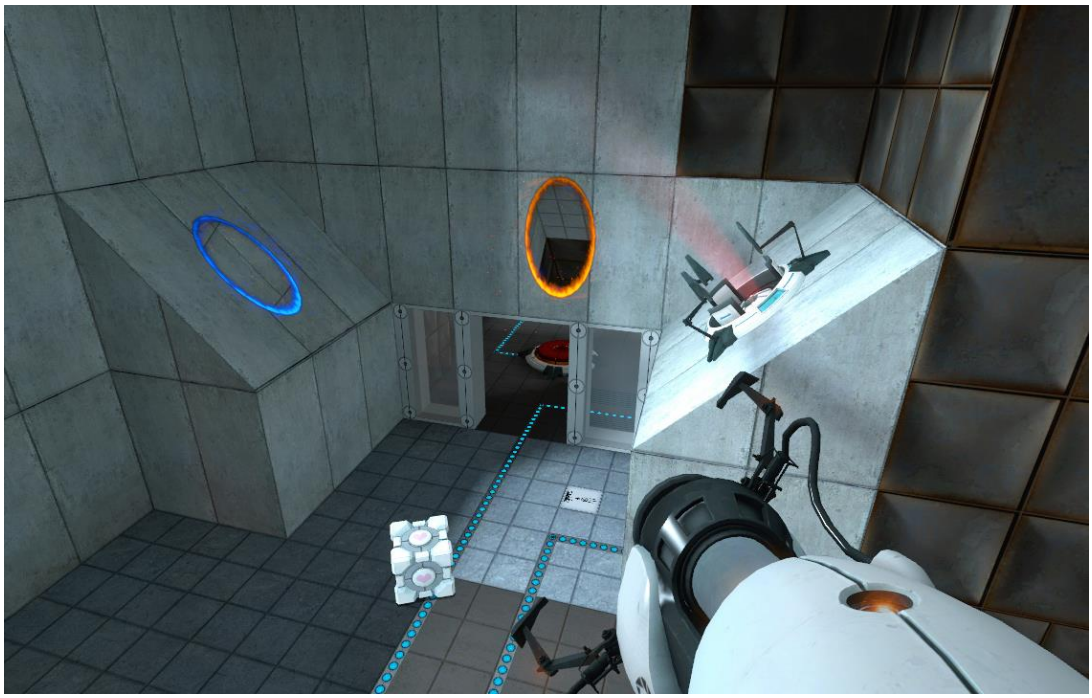


Figura 5. Imagen InGame de Portal.

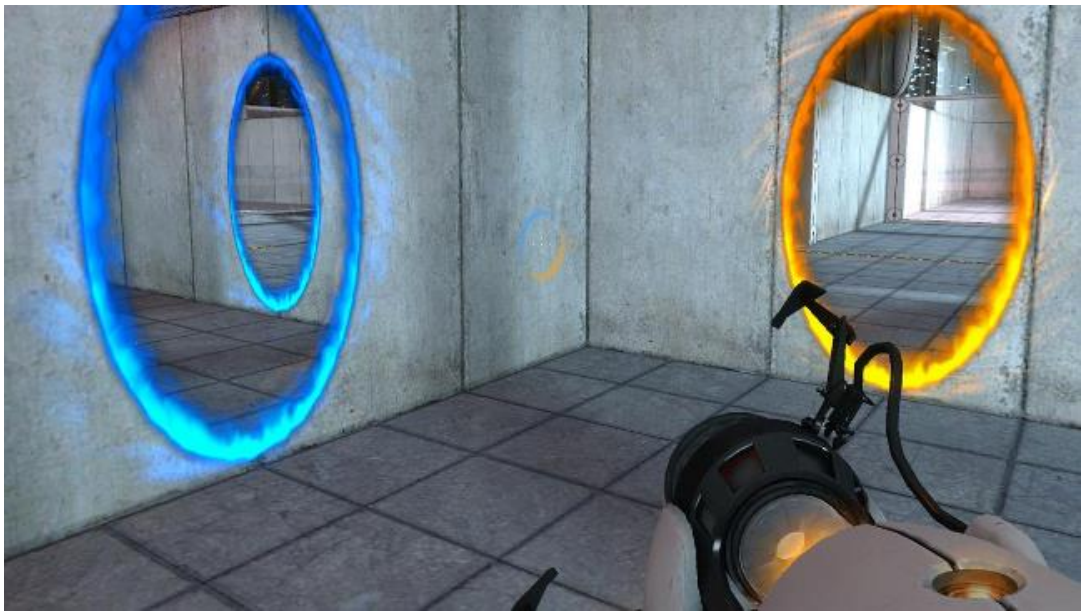


Figura 6. Imagen InGame de Portal.

Escenarios simples (aunque no así los niveles de dificultad) y niveles con soluciones aparentemente simples, como poner un objeto encima de un interruptor o accionar mecanismos, pero unas mecánicas innovadoras y entretenidas hacen de Portal uno de los videojuegos más destacados de los últimos años.

2.3.2. Ori and the Blind Forest

Ori es un juego plataformas 2D con mecánicas sencillas que se van introduciendo a lo largo de la historia. Mecánicas sencillas, salto, doble salto, escalada y rebote no parecen muy innovadores hoy en día, pero su gran punto a favor es el apartado gráfico y sonoro. Creando una ambientación que introduce al jugador en el juego, consigue potenciar enormemente el peso del juego.

Figura 7. Imagen InGame de Ori and the Blind Forest.

Figura 8. Imagen InGame de Ori and the Blind Forest.

Ori and the Blind Forest cuenta con más de 10 premios mundialmente reconocidos y 20 nominaciones, como Mejor Audio, Xbox Game of the Year y Best Art Direction (2015).

2.3.3. MAGICKA 2

Figura 9. Logo de Magicka 2.

Otro videojuego que me ha inspirado para ambientar este proyecto es Magicka. Un juego multijugador y divertido, con cientos de combinaciones, y sobretodo entretenido. El estilo de juego divertido y colorido ha inspirado en gran medida la ambientación de Twilight Whispers.

Figura 10. Imagen InGame de Magicka 2.

Para este proyecto he querido destacar la ambientación colorida y mágica, entremezclando diseños de videojuegos como Ori: The Blind Forest y Magicka 2.

2.3.4. WARCRAFT III: ELEMENTAL TOWER DEFENSE

Este modo de juego no fue un videojuego en sí, sino un mod que, aplicado al famoso RTS, nos permitía crear un mapa de jugabilidad Tower Defense en el que teníamos que evitar que ciertos enemigos llegaran hasta nuestra base, colocando estructuras defensivas. Además, uno de las principales novedades que incluía es que nos permitía manejar a un personaje, además de centrarnos únicamente en construir defensas.

Este mod, creado por los usuarios, popularizó bastante el género Tower Defense, que acabó dando un gran salto en PC con "Master Of Defense" en el año 2005.

Figura 11. Imagen InGame de Warcraft III: Reign Of Chaos.

Figura 12. Imagen InGame del personaje principal en Warcraft III: Reign Of Chaos, con mods.

3. OBJETIVOS

3.1. OBJETIVO PRINCIPAL DEL TRABAJO DE FIN DE GRADO

El objetivo principal de este proyecto es el diseño, creación y desarrollo de un videojuego 3D utilizando el motor gráfico Unity3D, con diversos añadidos técnicos y artísticos que den un acabado completo al producto final.

Por otro lado, se pretende conocer más a fondo el motor Unity, analizar sus características y aprender a adaptar cambios de forma iterativa, así como realizar el documento de diseño (GDD) del mismo.

3.2. DESGLOSE DE OBJETIVOS

A continuación se muestra una lista con los objetivos del proyecto de forma más específica por orden de trabajo.

- Desarrollar una idea realizable e implementable de un videojuego.
- Implementar todo el proyecto utilizando herramientas gratuitas y sin ningún tipo de presupuesto.
- Realizar el documento de diseño (GDD) del mismo.
- Planificar correctamente el desarrollo del proyecto.
- Aprender a organizar y gestionar el proyecto empleando metodologías de trabajo.
- Implementar las mecánicas del videojuego.
- Crear una interfaz de usuario (UI) de juego simple y llamativa.
- Conseguir un resultado atractivo visualmente para el jugador.
- Conseguir un nivel con apartados realistas en cuanto a mecánicas y apartado gráfico.
- Importar y utilizar correctamente el contenido descargable de la Asset Store de Unity.
- Implementar un código correcto, funcional, limpio y ampliable.

4. METODOLOGÍA

4.1. METODOLOGÍA DE DESARROLLO

Para la realización de este proyecto he optado por utilizar la metodología ágil FDD (Feature Driven Development), que se basa en la realización de un diseño preliminar con un objetivo final. Una vez planificadas todas las funcionalidades se iterará por todas ellas para ir completándolas.

El modelo FDD, al igual que otras metodologías, se enfoca en iteraciones cortas que entregan una funcionalidad tangible

Al no tener algunos aspectos totalmente definidos, se planteó la posibilidad de utilizar un Modelo Incremental que permitiera ir añadiendo funcionalidades a medida que avanzara el proyecto, pero finalmente se decidió continuar con la metodología FDD dado el diseño preliminar del proyecto al cobrar firmeza.

4.2. HERRAMIENTAS Y GESTIÓN

- *Unity 5.6*

Como ya detallamos antes, Unity será el motor de juego para el desarrollo del proyecto, y la plataforma objetivo sistemas de sobremesa.

- *Visual Studio*

IDE utilizado para trabajar sobre C#. Presenta algunas ventajas sobre MonoDevelop (IDE por defecto en Unity 5). Personalmente considero que presenta algunas ventajas interesantes con respecto a depuración de código y es un entorno ya conocido.

- *C#*

Lenguaje de programación. Se ha elegido en lugar de javascript por su parecido con C++, por el conocimiento previo sobre el mismo y por la magnitud de soporte en línea en foros y ayudas que se puede encontrar relacionado con Unity.

- *Gimp*

Software de diseño utilizado para diseñar y crear la interfaz de usuario. Se ha elegido en lugar de Photoshop por su similitud, amplias posibilidades y por contar con una versión gratuíta, para adecuarse al objetivo de presupuesto nulo.

- *Google Drive*

Plataforma de almacenamiento. Empleada para documentación y seguimiento.

- *Blender*

Software multiplataforma destinado al modelado, iluminación y renderizado. En este proyecto se empleará para retocar ciertos modelos de los personajes y crear algún contenido básico, ya que el diseño artístico propio no es el principal propósito del mismo.

Al igual que con la plataforma de diseño y preferir otros softwares como 3DstudioMax, Blender es un software gratuito.

- *Trello*

Figura 13. Logotipo de Trello

La herramienta empleada para el seguimiento del proyecto ha sido Trello, ya que es una herramienta flexible y muy minimalista que permite separar las funcionalidades diseñadas en “por hacer”, “en progreso” y “terminadas”

4.3. CONTROL DE VERSIONES

- *Sourcetree/Git*

El control de versiones se ha realizado utilizando Github, mediante el software Sourcetree, ya que es un entorno de trabajo conocido y muy completo. A pesar de ello, las versiones subidas al repositorio (push) han sido enviadas en versiones estables, no por cada cambio o contenido añadido.

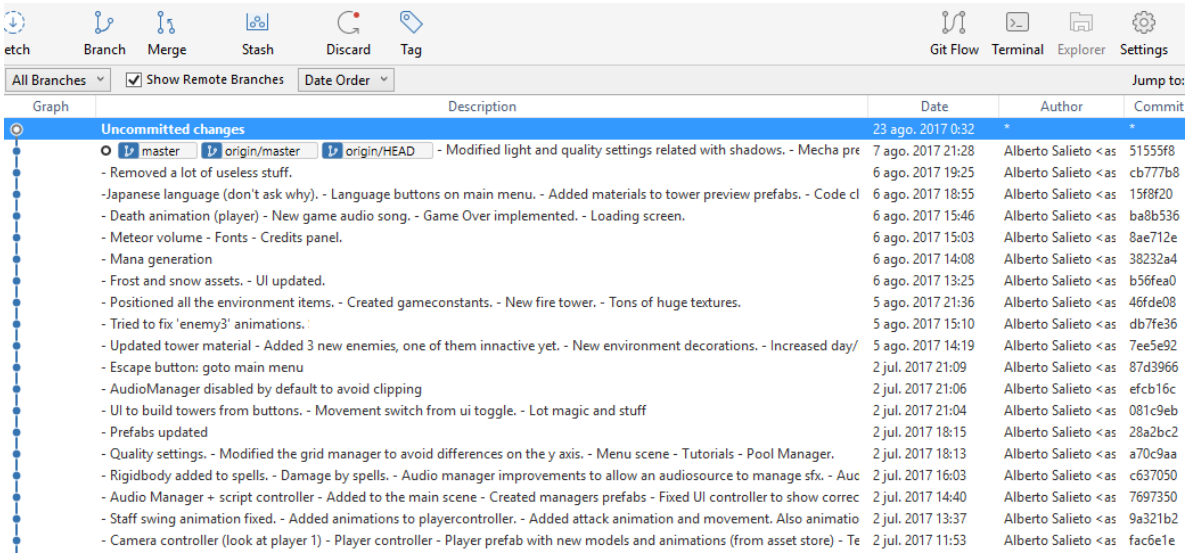


Figura 14. Repositorio del proyecto en el entorno de Sourcetree.

Los elementos contenidos en el repositorio incluyen código fuente, modelos, assets y contenido, no encontrándose pues en él documentación, seguimiento del proyecto o ejecutables finales, principalmente por temas de privacidad al ser un repositorio público.

5. CUERPO DEL TRABAJO

En este apartado se especificará detalladamente el trabajo desarrollado, dividido en dos bloques principales:

- Un primer bloque, destinada a la elaboración, desarrollo y explicación del Game Design Document (GDD) o documento de diseño del videojuego, donde se explican los detalles, especificaciones y aspectos de diseño del videojuego, así como herramientas y funcionalidades.
- Un segundo bloque correspondiente a los aspectos técnicos en profundidad e implementación del videojuego. Dada que esta es una parte prácticamente técnica, se acompañarán de capturas y ayudas visuales que ayuden a comprender el desarrollo.

5.1. GAME DESIGN DOCUMENT

En este bloque se especificarán todos los aspectos de diseño y funcionalidades del proyecto. Se trata, pues, de un guión a seguir a medida que se desarrolla. El nombre elegido para el proyecto y videojuego es Twilight Whispers.

Figura 15. Icono del videojuego

5.1.1. TÉRMINOS GENERALES

Figura 16. Logo del videojuego

5.1.1.1. ARGUMENTO

Ambientado en una época fantástica-medieval, encarnamos a una hechicera protectora del bosque. Desafortunadamente, los seres de la oscuridad acechan desde las sombras no nos lo pondrán nada fácil…

Durante la partida, tendremos que impedir que los seres de la noche alcancen los portales que comunican las ciudades de nuestro reino, construyendo estructuras mágicas y realizando encantamientos para retrasar su invasión el mayor tiempo posible.

5.1.1.2. GÉNERO

El género del videojuego es Tower Defense – Estrategia.

"Tower Defense es un subgénero de los videojuegos de estrategia en tiempo real. El objetivo es lograr que las unidades enemigas no lleguen a cruzar el mapa, para lograrlo se deben construir torres que las atacan al pasar. Tanto los enemigos como las torres tienen diferentes habilidades y costes."

A pesar de ello, también controlaremos a una heroína que cuenta con varios encantamientos de defensa. La elección apropiada de torres y su ubicación es la estrategia esencial, ya que nos encontraremos con enemigos más rápidos, lentos o resistentes.

El principal referente del género ha sido el escenario Tower Defense del antes mencionado videojuego Warcraft III.

5.1.1.3. AUDIENCIA

En el ámbito de audiencia, consideraré el target ideal al que se destina el juego: los principales consumidores del mismo, bien por gusto, género o atractivo. Por otro lado, la clasificación PEGI del videojuego viene clasificada por los diferentes aspectos que pueda contener el videojuego, coincida o no con el target deseado.

Twilight Whispers es un juego orientado a un público bastante general, desde los 8~18 años, dado que requiere cierta estrategia para encontrar la mejor manera de posicionar las torres para conseguir una defensa sólida. De otra manera, puede resultar frustrante para el jugador no tener claro que la mejor opción es centrarse en defensas en lugar de atacar a rienda suelta a los enemigos.

5.1.1.4. CLASIFICACIÓN PEGI

La clasificación PEGI alude principalmente al contenido violento, lenguaje soez o sexual de un videojuego. Algunos elementos son bastante subjetivos, como la violencia o escenas que pueden resultar desagradables, sobre todo en los límites entre una edad u otra menor de 18.

En el caso de este videojuego en concreto, no considero que contenga contenido de este tipo. A pesar de ello, por mostrar algunos enemigos de apariencia humanoide desfigurada, lo he establecido en PEGI-12, que se define con la siguiente descripción:

"Videogames that show violence of a slightly more graphic nature towards fantasy character and/or non graphic violence towards human-looking characters or recognisable animals, as well as videogames that show nudity of a slightly more graphic nature would fall in this age category. Any bad language in this category must be mild and fall short of sexual expletives."

Además, dada la comparativa realizada con otros títulos, probablemente se introduciría la descripción adicional de Violencia

Figura 17. Iconos PEGI

5.1.1.5. APARIENCIA

La apariencia será un aspecto muy destacable del juego, ya que uno de los objetivos principales del proyecto es conseguir crear una gran ambientación fantástica. Trataremos de utilizar un ambiente capaz de inmerjirnos en un escenario de fantasía, habitado por criaturas mágicas y azotado por malvados seres de la noche.

Para la parte mágica, hechizos y elementos beneficiosos que se asocien a la "bondad", elementos a proteger y ayudar al jugador, se utilizarán siempre elementos claros y luminosos, siempre asociados a factores brillantes y mágicos.

Figura 18. Referencia visual de elementos mágicos dentro de un bosque. Fuente: Devianart

En cuanto al bando de los enemigos, la parte “maligna”, se utilizarán elementos tenues y oscuros, privados de luminosidad y que cargen negativamente el ambiente para dar la sensación de necesidad de erradicarlos, y que motive al jugador a defenderse de ellos.

Para conseguir estos resultados, se utilizará muy frecuentemente sistemas de partículas. Cada elemento de cada una de las partes, malvada y bondadosa, compartirá características comunes que definirán muy claramente lo beneficioso de lo perjudicial.

Figura 19. Referencia visual de elementos negativos dentro del entorno mágico. Fuente: Ori

A grandes rasgos, elementos como las hadas, hechizos, torres o portales deberán ser muy brillantes, mientras que guardaremos colores, texturas y partículas más oscuras para enemigos.

Otro elemento muy importante que refuerza estos conceptos de luz-oscuridad la encontraremos en la progresión de la partida- Los enemigos aparecerán durante la noche, momento en el que la iluminación global de la escena se atenuará hasta que llegue el alba, momento en el que la luz acabará con los enemigos restantes.

5.1.1.6. OBJETIVO DEL JUEGO

El objetivo del juego es aguantar el máximo tiempo posible sin que los enemigos alcancen 10 portales de teletransporte, que irán apareciendo en el nivel a medida que avanza la partida.

5.1.1.7. Características y objetivos

- Gran aspecto visual. La mayoría de elementos deberán basarse en aspectos relacionados con la iluminación, sombras, brillos y reflejos especulares. Las partículas desempeñarán un gran papel para dar esta ambientación.
- Controles sencillos. Podrá jugarse tan sólo con el ratón.
- Sistema de localización e idiomas.
- Sistema totalmente parametrizable, listo para añadir contenido.

5.1.1.8. Resumen del flujo de juego

- Amanece.
- Aparecen portales por todo el nivel.
- El jugador coloca torres de defensa.
- Anochece.
- El jugador ya no puede construir más torres. En su lugar, puede lanzar hechizos.
- Cada cierto tiempo, aparecen enemigos en los diferentes puntos de spawn del nivel.
- Si un portal es alcanzado, se reduce la puntuación.
- Termina la noche.
- Los enemigos que han sobrevivido mueren automáticamente.
- Vuelve a amanecer. Se repite el flujo hasta que los enemigos alcancen 10 portales.

5.1.2. Mecánicas de juego

5.1.2.1. Progresión

Durante la partida, habrá un flujo de día-noche. Al amanecer, será cuando aparezcan los portales. Durante el día, el jugador se encargará de construir torres defensivas. De este modo, el jugador tendrá

que ir moviéndose por el mapa y no centrarse sólo en poblar una única zona de defensas, sino repartirlas por el escenario, ya que los portales se generarán en una posición diferente al amanecer.

5.1.2.2. CONTROLES

Los principales controles del juego utilizarán los botones derecho e izquierdo del ratón.

5.1.2.2.1. MOVIMIENTO

El personaje se controlará con el click izquierdo del ratón, utilizando la mecánica click-to-move, pero el personaje no se moverá automáticamente hasta la posición destino, sino que avanzará en esa dirección. Dado que la velocidad de movimiento es un factor muy importante en el juego, no tiene sentido facilitar este aspecto calculando un algoritmo de camino más corto A* o navmesh.

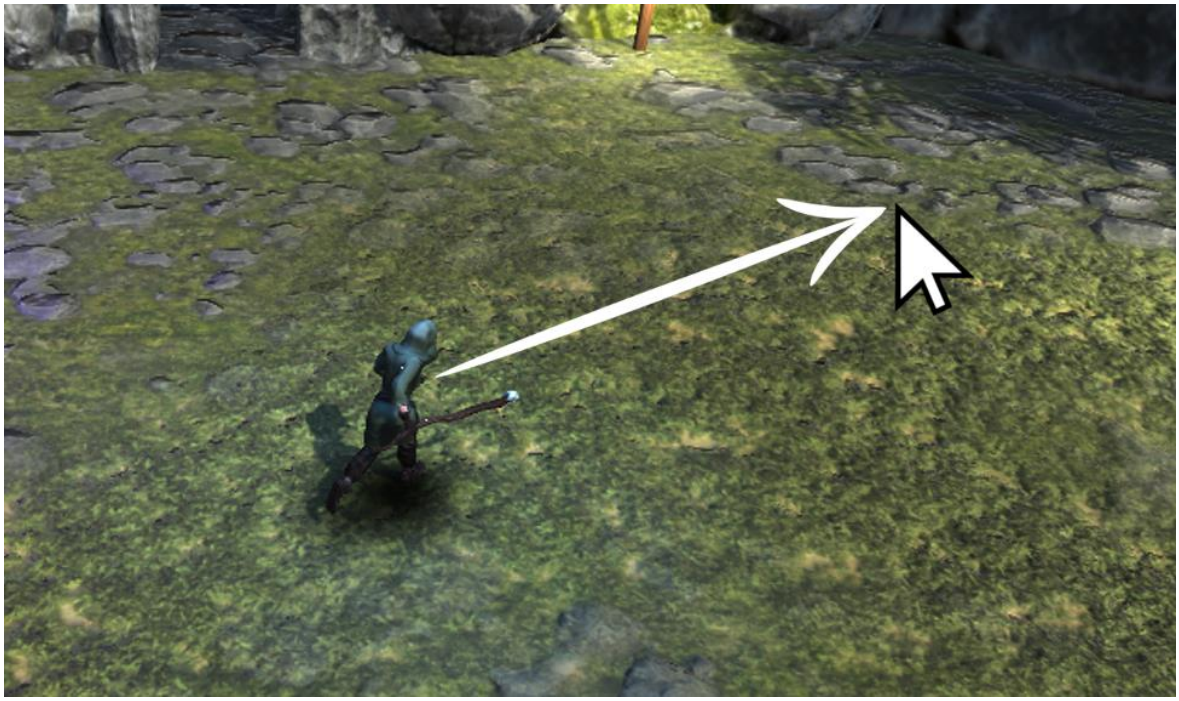


Figura 20. Referencia visual sobre cómo será el movimiento. Fuente: propia

Por este motivo el personaje se moverá hacia la posición del ratón mientras se mantenga el click izquierdo, a pesar de encontrar colisión con un enemigo o una torre.

5.1.2.2.2. ACCIONES

- Acciones de noche
    - Atacar

Con el botón derecho del ratón podremos lanzar un hechizo ofensivo: un proyectil helado. Este no causa un gran daño, pero es bastante efectivo contra rematar enemigos con baja salud. El proyectil avanzará hacia la dirección del puntero hasta que colisione con algún enemigo o se salga de los límites del escenario.

Lanzar este proyectil no tiene coste.

    - Lanzar hechizo

También contamos con un ataque demoledor basado en una lluvia de meteoritos. Este ataque se desata desde el panel lateral cuando se ha reunido suficiente maná y la barra de carga está al máximo.

Utilizar este ataque consumirá todo el maná acumulado y desatará una lluvia de meteoritos sobre el escenario, dañando a los enemigos que encuentre a su paso.

- Acciones de día
    - Construir torres

Durante el día estará disponible la opción de construir torres. Con el botón izquierdo del ratón, podremos seleccionar la torre que queramos construir en el panel izquierdo. Una vez seleccionado, aparecerá una silueta de la torre a colocar allá donde apuntemos con el ratón.

    - Atacar

Durante el día también podremos atacar. Puede parecer que no tiene mucha utilidad atacar por el día ya que no aparecen enemigos, pero puede emplearse para estudiar algunas posiciones estratégicas y prepararse para la oleada enemiga.

5.1.3. PERSONAJES

5.1.3.1. LISZU

Liszu es el personaje principal de la historia. Se trata de una hechicera que habita en el bosque oscuro, lugar donde aparecerán los seres de la noche que tratarán de invadir el reino.

Viste una túnica con capucha y porta una vara mágica gracias a la que es capaz de realizar hechizos. Se volvió protectora del bosque gracias a sus facultades innatas en la hechicería.

Figura 21. Personaje principal, Lizsu.

A continuación se especifican el daño que causa cada ataque.

	Daño	Coste de maná	Tiempo de ejecución
Proyectil helado	25	-	1.5
Lluvia de meteoritos	100 (por meteorito, ataque en área)	100	2.5

Figura 22. Características del personaje principal.

5.1.3.2. GÓLEM

Los gólems son un tipo de enemigo con rol de tanque. Esto es, que posee mucha vida, pero su avance es lento en comparación con los otros enemigos.

Figura 23. Enemigo Gólem.

Es un ser con apariencia robusta, grande y musculoso, y con aspecto de confiar más en la fuerza bruta que en la inteligencia, por lo que necesitarán más tiempo para pensar qué hacer cuando su objetivo desaparezca y tengan que buscar el siguiente portal más cercano.

5.1.3.3. OGRO

Los ogros son criaturas malvadas, de un aspecto más oscuro que los gólems pero de menor tamaño. Esto les permite avanzar más deprisa que ellos, pero carecen de la dureza propia de los gólems.

Figura 24. Enemigo Ogro.

La falta de fuerza y aguante la compensan, pues, con algo de velocidad tanto motriz como intelectual.

5.1.3.4. ESQUELETO

Los esqueletos son seres aberrantes que no dudarán en abalanzarse sobre los portales mágicos. Visten harapos y se mueven a gran velocidad, siendo así más fácil acertarles. Sin embargo, no cuentan apenas vida, pero resultan una molestia a la hora de acertar los hechizos. Además, cuentan con unos rápidos reflejos que les permite localizar rápidamente los portales más cercanos.

Figura 25. Enemigo Esqueleto.

Los esqueletos no son un problema a la hora de acabar con ellos, pero resultan bastante molestos ya que son capaces de alcanzar rápidamente los portales del jugador.

A continuación se describen detalladamente de cada tipo de enemigo:

	Velocidad	Vida	Reflejos
Liszu (referencia)	20	-	-
Gólem	12	100	60
Ogro	18	80	75
Esqueleto	22	20	100

Figura 26. Características de cada personaje.

5.1.4. TORREONES

Contaremos con 3 tipos de torres defensivas, que podremos colocar dependiendo de nuestro estilo de juego o estrategia. Cada tipo de torre cuenta con unas especificaciones que se recopilan en la tabla de abajo.

- Torre mágica

La torre mágica (o torre azul) disparará proyectiles mágicos a los enemigos que pasen por su lado. Todos los proyectiles acaban impactando en sus objetivos, pero los de esta torre son algo más lentos que las otras torres, pero con un poder de ataque medio. Su rango también está en la media de todas las torres.

Figura 27. Torre mágica.

Estas características las hacen ideales para colocarlas en las inmediaciones de las zonas más transitadas, pero en una segunda línea de defensa, para aprovechar su largo alcance a pesar de no causar un gran daño.

En cuanto a su estilo, mantiene en su parte superior un orbe de brillo azulado, que al ser disparado dispara un proyectil similar al objetivo.

- Torre láser

La torre láser (o torre verde) son muchísimo más potentes que las torres mágicas convencionales. Su cadencia de ataque también es mayor, pero a cambio cuenta con un rango de ataque mucho más reducido.

Por esto mismo son ideales para poblar las vías de entrada de los enemigos. Son una sólida defensa en primera línea, pero se vuelven menos útiles a medidas que se ensancha el escenario, ya que sólo son útiles si los enemigos se acercan a ellas. Dado que los portales varían su posición, puede ser un desperdicio ocupar casillas lejanas con estas.

Figura 28. Torre láser.

Su estilo visual es el más característico, ya que impacta automáticamente en el objetivo. Desde la parte superior de la torre dispara un rayo de intenso brillo verdoso directamente a sus objetivos, que no tarda en desaparecer.

- Torre ígnea

La torre ígnea (o torre roja) tiene un ataque ardiente continuado, con mucha velocidad de disparo, pero bajo daño. Su alcance es bastante grande, por lo que es perfecto para rematar unidades que se hayan escapado con poca vida y así conseguir que las otras torres más potentes se encarguen de los siguientes enemigos. Por esta razón, esta torre suele situarse detrás de todas las demás, y es muy eficiente para acabar con enemigos veloces como los esqueletos.

Figura 29. Torre ígnea

Visualmente, los disparos de la torre ígnea son un proyectil envuelto en llamas. Dado que tiene una cadencia de fuego muy alta, se parecerá más a un lanzallamas que a proyectiles individuales.

	Cadencia de fuego	Velocidad de proyectil	Daño
Torre mágica	1.5	2	30
Torre láser	1	100	80
Torre ígnea	3	10	15

Figura 30. Tabla con las especificaciones de cada estructura defensiva.

5.1.5. PORTALES

Los portales son los puntos de acceso rápido a zonas emblemáticas de nuestro reino. Debemos evitar por todos los medios que los enemigos lleguen a estos puntos, por lo que el juego terminará si esto ocurre un número determinado de veces.

Estos portales aparecen al comienzo del día en zonas aleatorias del escenario, siempre y cuando la casilla no esté ya ocupada por una torre. Desaparecen cuando son alcanzados por un enemigo o al llegar el alba. No tienen interacción con el usuario, por lo que puede atravesarse sin ningún tipo de colisión.

Figura 31. Portal

Visualmente, desprenden una esencia mágica a su alrededor, así como luz propia,y se crean y se destruye mediante una sencilla animación.

5.1.6. ESCENARIO

El escenario se compone de un nivel limitado por elementos naturales, como vallas o acantilados, que impiden que el jugador se salga del nivel. El jugador tiene libertad para moverse dentro de estos límites, al igual que los enemigos. No sigue pues restricciones como las que podría establecer un tablero por casillas.

A pesar del libre movimiento de los personajes, de forma invisible se establece un tablero hexagonal que servirá de base para la colocación de torres. Este tablero estará completamente vacío al principio y se irá llenando a medida que se coloquen las torres. No se representará visualmente de forma directa, pero podrá tenerse presencia del mismo en el modo de construcción de torres.

Dado que las torres sí que se posicionan ancladas a estas casillas hexagonales, en el modo de creación se posicionarán en la casilla correspondiente, por lo que no será posible dejar una torre entre casillas.

Otra restricción sobre las estructuras defensivas recae sobre las casillas adyacentes, que no podrán ser ocupadas por otro torreón.

Sin embargo, toda casilla no ocupada por una torre puede ser punto de aparición de un portal.

5.1.6.1. NIVEL DE JUEGO: CLARO DEL BOSQUE

El nivel principal transcurre en el Bosque Oculto. Se trata de un misterioso claro rodeado por 3 cuevas, de las que emergen los seres de la oscuridad: una superior y dos laterales, una a cada lado.

En la parte inferior hay posicionadas unas vallas decorativas que restringen la salida del personaje del escenario, en lugar de los acantilados, que molestarían en la pantalla en algunos planos de cámara.

En cuanto a la ambientación, aparece una suave niebla a medida que se acerca la noche, y los elementos de decoración resaltan por la vegetación y sencillez, dando la sensación de estar en el claro del bosque.

Figura 32. Escenario del claro del bosque.

5.1.7. INTERFAZ (UI)

En este apartado se detallarán los aspectos de diseño de toda interfaz 2D

5.1.7.1. HUD

El HUD hace mención a la interfaz 2D (interactuable o no) dentro del juego durante la partida. En el nivel de juego será necesario mostrar diferentes apartados.

- Menú superior

Figura 33. Marcador superior

- Temporizador y etapa actual

En la parte superior de la pantalla se establecerá un temporizador, que irá disminuyendo a medida que pase el tiempo. Este tiempo marcará el tiempo restante para que llegue la siguiente etapa en formato de MM:SS, acompañado por la siguiente etapa (día/noche)

- Portales restantes

Número de portales restantes para que termine la partida. El número inicial será 10, e irá disminuyendo cada vez que un enemigo alcance uno. Si esta puntuación llega a 0, terminará la partida.

- Oleadas superadas

Es la puntación del jugador. Cada ciclo cuenta como un punto, por lo que aumentará al final del mismo. No tiene valor máximo.

- Menú lateral

- Torreones disponibles

Dispondremos de un botón para cada uno de los tipos de torre (3). Desde aquí, durante el día, se podrán construir las estructuras. Durante la noche, estarán deshabilitados y con alguna marca que lo muestre visualmente.

- Maná

El maná se genera automáticamente con el tiempo o bien golpeando a los enemigos con el hechizo básico, por lo que la mejor forma de representarlo visualmente es una barra de carga. Esta barra será una barra vertical que se llenará de un color azul, generalmente asociado con el color del maná.

- Lanzar hechizo

Este botón estará generalmente deshabilitado, a no ser que la barra de maná esté llena. Cuando esto ocurra, el botón se activará, permitiendo desencadenar una lluvia de meteoritos y consumiendo todo el maná. Cuando esto ocurra, el botón volverá a deshabilitarse, adquiriendo un tono grisáceo.

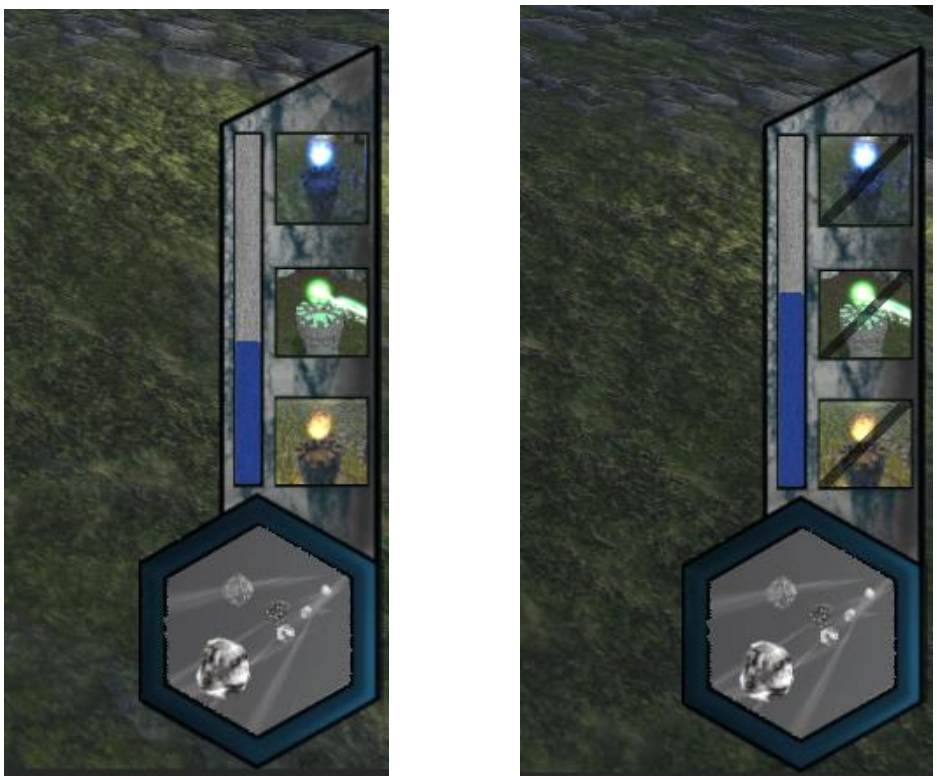


Figura 34. A la izquierda, menú lateral con los torreones disponibles. A la derecha, bloqueados.

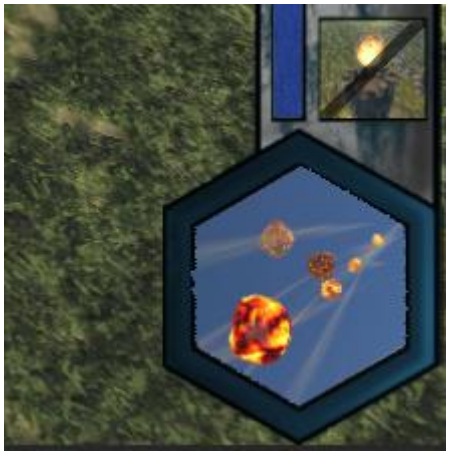


Figura 35. Hechizo final disponible.

5.1.7.2. MENÚS

5.1.7.2.1. MENÚ PRINCIPAL

Menú con opciones de juego, así como el título del juego. El menú principal mostrará las siguientes opciones:

- Nueva partida: Comienza un nuevo juego.
- Opciones: Idioma del juego y créditos
- Salir: Cierra la aplicación.

En cuando a la decoración trasera, se mostrará un escenario preparado para ello, en que aparecerá Liszu, el personaje principal, junto a algunos elementos decorativos o efectos.

Figura 36. Menú principal de juego.

Figura 37. Menú de opciones.

5.1.7.2.2. MENÚ PARTIDA TERMINADA

Cuando termine la partida, el personaje principal realizará la animación de muerte, y en el espacio del temporizador aparecerá un texto de Fin de partida.

Además, aparecerá un botón en la parte inferior de la pantalla para reiniciar la partida.

5.1.8. CÁMARA

La cámara del juego es una cámara en 3º persona sobre el jugador. El ángulo y lejanía de la cámara es fija, de unos 35º-55º, como suele ser común en los juegos RTS como Starcraft o Warcraft.

5.1.9. IA

Los enemigos aparecerán en los puntos de spawn. Desde esa posición, encontrarán el portal más cercano y encontrarán la manera más rápida de llegar hasta él. Evitarán, pues, las colisiones de los torreones, pero no realizarán cambios en su trayecto para buscar un camino menos peligroso. Como

en la mayoría de juegos de género tower defense, la jugabilidad recae sobre la cantidad de enemigos que aparecerán en el escenario, no en su habilidad para evitar las estructuras defensivas.

Por otro lado, en este escenario aparecen varios objetivos o portales. Dado que la ronda no termina cuando uno es alcanzado, los enemigos que tenían un objetivo que ya ha sido alcanzado buscan el siguiente portal más cercano.

Aquí entra el juego el valor de cada tipo de enemigo que llamamos "reflejos". Cada uno tiene un tiempo de espera desde que pierde su objetivo (el antiguo portal objetivo) hasta que encuentra el siguiente. Durante este tiempo, el enemigo se queda inmóvil, y pasado este tiempo se dirige al nuevo objetivo.

5.1.10. Música y audio

Se utilizarán 2 pistas de audio para el juego, una de ellas para el menú principal y otra para la partida en curso. Ambas recrearán un ambiente de tensión acorde con la ambientación del juego.

5.1.10.1. Efectos de sonido

1. Torre mágica (azul).
2. Torre láser (verde).
3. Torre ígnea (roja).
4. Hechizo básico.
5. Hechizo final.
6. Portal.
7. Spawn enemigo.
8. Click de botón.

5.1.11. Lenguaje

El juego es capaz de cambiar de lenguaje en ejecución. Además, dispone de un sistema preparado para cambiar de forma sencilla texto e incluir nuevos idiomas.

Los idiomas disponibles son inglés, español y japonés.

5.1.12. Guía Técnica

Como se mencionó en el apartado de las estructuras defensivas, es muy importante saber distribuir nuestras defensas. Es indispensable cubrir las entradas de los enemigos con torres, y una buena estrategia es cubrirlas con torres láser. No cuentan con mucho rango, pero en las entradas podemos asegurar que siempre darán en el blanco, ya que es una parte segura de la ruta de los enemigos. A partir de ahí, necesitaremos cubrir los demás huecos con torres mágicas y cubrir los centros con torres ígneas. Pero, ¿por qué, si estas son las menos poderosas?

Enemigos como los esqueletos cuentan con una gran velocidad, y pueden salir airosos, aunque con poca vida, de nuestras defensas principales. Rematarlos será nuestro trabajo con los hechizos básicos, pero siempre podremos ayudarnos de las torres de mayor alcance.

Algo que siempre debemos tener en cuenta es que no importa el estado en el que llegue el enemigo al portal, ya sea uno o varios, con poca vida o dañados, lo perderemos igualmente, por lo que no hay que tomarse a la ligera a ninguno de ellos y cubrirnos bien las espaldas.

En cuanto al ataque final, siempre es una buena solución para limpiar el nivel, pero deberemos tener en cuenta que no es un ataque dirigido, sino de área, y que también podemos ser afectados por él: no en forma de daño, pero puede desplazarnos si un meteorito colisiona cerca de nosotros.

5.2. DESARROLLO E IMPLEMENTACIÓN

5.2.1. ESTRUCTURA GENERAL

5.2.1.1. PATRONES DE DISEÑO

Para los controladores (managers) de juego, se utilizará el patrón de diseño Singleton. Este patrón consiste en una clase que sólo puede ser instanciada una única vez en la escena, y permite ser accesible desde cualquier otro script. Esto se realiza comprobando si ya ha sido instanciada previamente cuando se solicita acceso a sus métodos. Si ya lo ha sido, devuelve la instancia de la misma para efectuar lo que se necesite, y si no existía previamente, crea una nueva y la devuelve.

```
1.  using UnityEngine;
2.
3.  public abstract class Singleton < T > : MonoBehaviour where T: class, new()
4.  {
5.      public static T Instance
6.      {
7.          get
8.          {
9.              return sInstance;
10.         }
11.     }
12.
13.     public virtual void Awake()
14.     {
15.         DontDestroyOnLoad(gameObject);
16.         if (sInstance == null)
17.         {
18.             sInstance = this as T;
19.         }
20.         else
21.         {
22.             Debug.Log(name + ": Error: already initialized");
23.         }
24.     }
25.     private static T sInstance = null;
26. }
```

Figura 38. Patrón Singleton.

A pesar de todas las ventajas que ofrece este patrón, no conviene abusar del mismo, al igual que de las variables públicas, ya que deforma el diseño básico de toda la estructura. Tener una forma "fácil" de acceder globalmente a tanto variables como métodos puede llevar a fallos en la estructura y crear dependencia a la misma.

5.2.1.2. ESTRUCTURA DE DATOS: CLASES

Crearemos las clases básicas del juego, que serán las siguientes:

- PoolManager: Será el encargado de instanciar los objetos del juego de forma predeterminada.
- LevelManager: Donde estableceremos las reglas del juego: duración del ciclo día/noche, portales disponibles, tiempos de juego...
- LocalizationManager: El encargado de extraer los textos localizados de los archivos de idiomas.
- AudioManager: Controlador principal de audio y sonidos de juego.
- GridManager: El manager del grid de torres y portales. Controla las casillas, su ocupación y separación, etc.
- TowerManager: Un controlador básico donde están añadidas las torres que se pueden colocar, parametrizadas, para poder editar fácilmente cuales están disponibles y en qué orden.
- UIManager: Manager de la interfaz de juego: menú lateral (construcción de torres), tutorial
- PlayerManager: Es el encargado de gestionar los parámetros principales del personaje principal.
- Enemy: Controlador con las características de los enemigos: tipo, vida máxima y actual, velocidad...

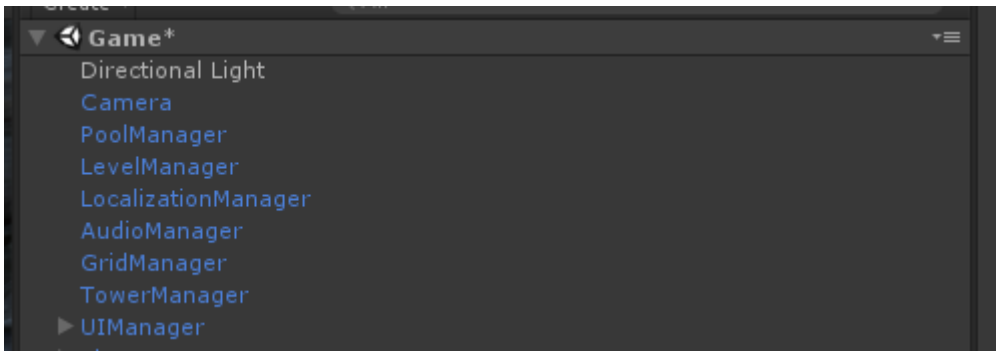


Figura 39. Principales clases Manager del juego.

5.2.1.3. ESTRUCTURA DE DATOS: VARIABLES.

Crearemos las clases básicas En cada clase se establecerán las variables privadas. Aquellas que sean de ámbito configurable, como velocidad de enemigos, vida, costes, etc., serán de estilo serializable. En Unity, se configura añadiendo el campo [SearizableField] encima de la declaración de la variable. Con esto, el parámetro aparecerá en el inspector lateral de Unity para ser configurado manualmente.

```
1.  [SerializeField]
2.  private float _attackDelay = 0.5 f;
3.
4.  [SerializeField]
5.  private float _bulletSpeed = 10 f;
6.
7.  [SerializeField]
8.  private float _bulletDamage = 7 f;
9.
10. [SerializeField]
11. private GameObject m_spell;
```

Figura 40. Ejemplo de serialización.

5.2.2. ESCENARIO

El escenario viene limitado por elementos decorativos que cercan el espacio principal. Además, cuenta con un grid invisible que se utilizará para colocar las estructuras defensivas.

Figura 41. Escenario principal

Las paredes son assets de diferentes tamaños, combinados para crear las paredes. Entre estos assets se encuentran rocas, acantilados y cuevas.

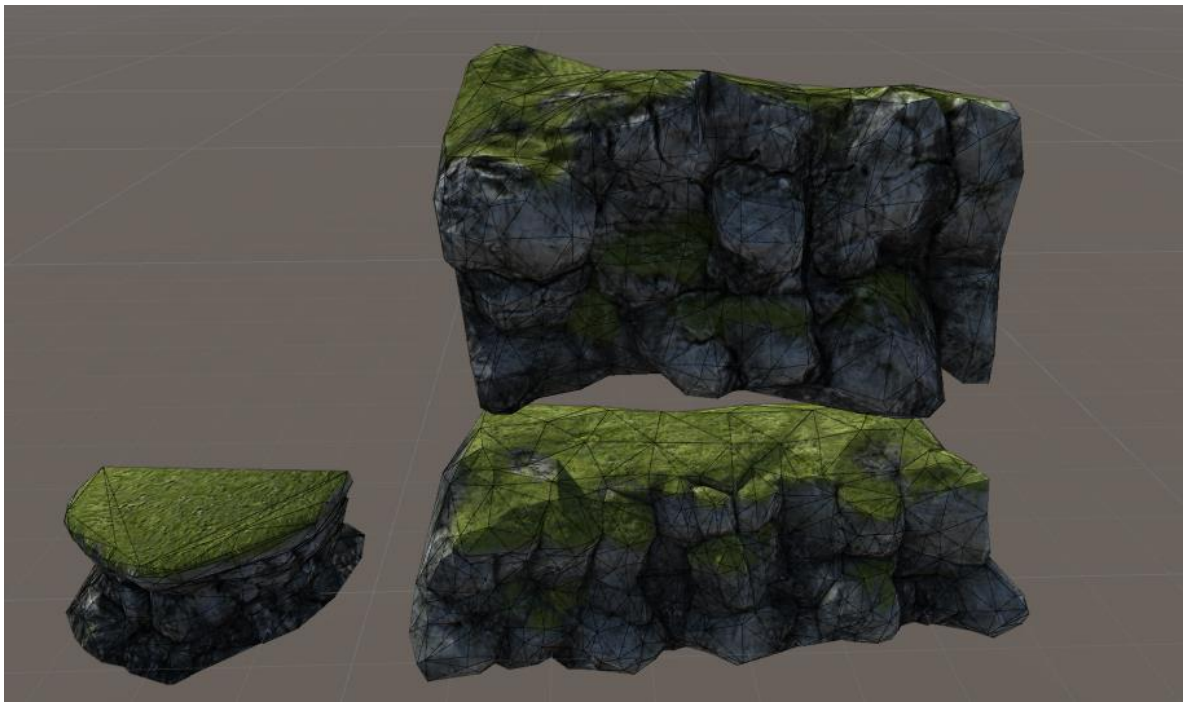


Figura 42. Acantilados.

Las decoraciones incluyen rocas y árboles, principalmente.

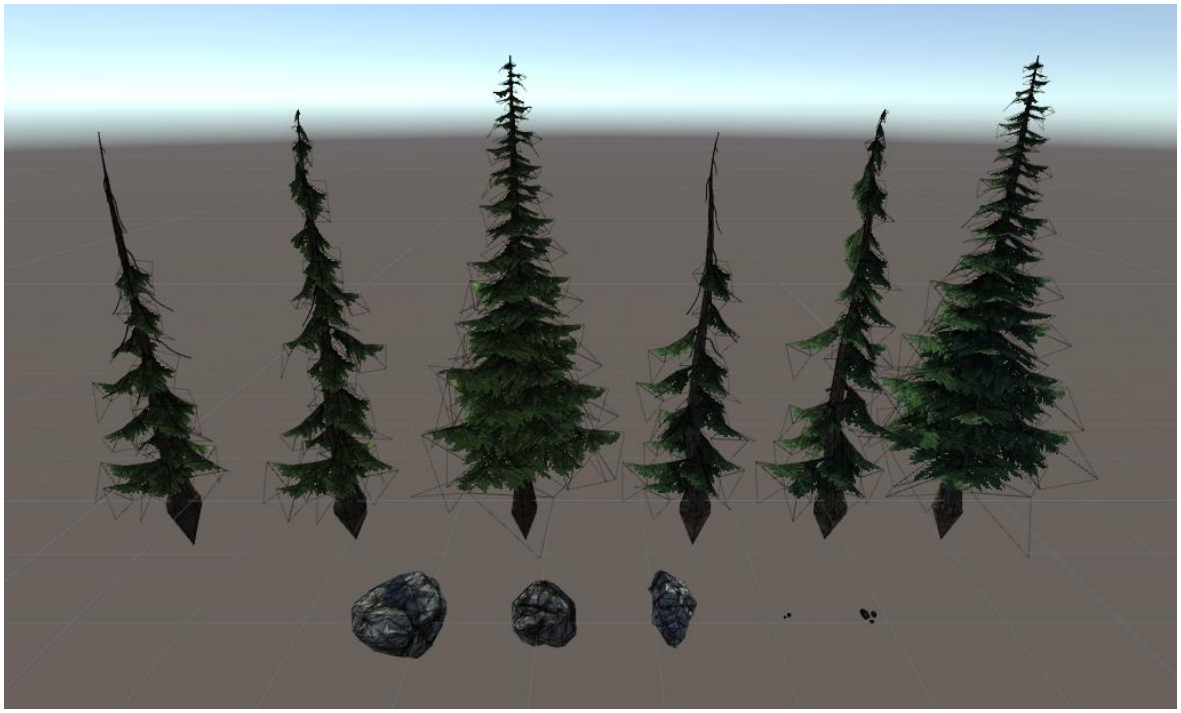


Figura 43. Árboles y otras decoraciones.

5.2.2.1. SISTEMA DE POSICIONAMIENTO DE TORRES

Estableceremos las torres en una matriz hexagonal. Para ello, definiremos la la clase MyTile, que contendrá la información de cada casilla, así como comprobar si está ocupada por un portal o una torre. Luego, generaremos la matriz de MyTiles, definida como MyGrid.

5.2.2.1.1. TILE

Para cada casilla deberemos calcular las coordenadas de los centros y los vértices de cada uno. Una matriz hexagonal puede tener diferentes configuraciones. Una característica es la orientación de las casillas, que puede configurarse de las siguientes maneras:

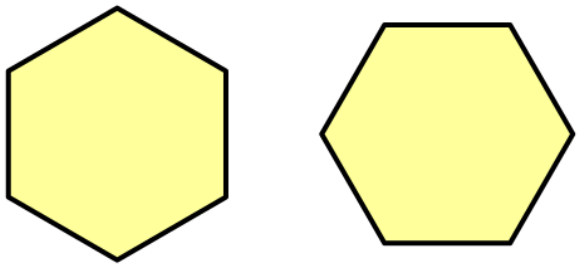


Figura 44. Orientaciones posibles

Otro valor necesario para calcular la separación entre casillas es el ancho y alto de la misma. Para facilitar este cálculo, utilizaremos un modelo de un bloque hexagonal, que a posteriori estará oculto.

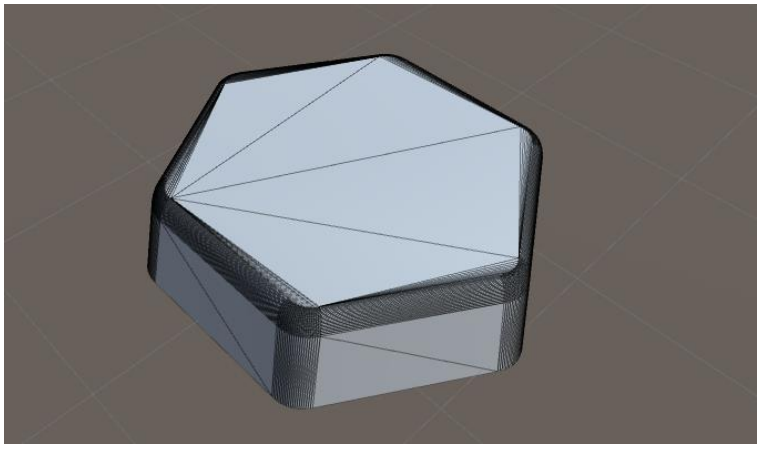


Figura 45. Casilla del tablero.

Otro aspecto a tener en cuenta a la hora de generar el tablero es el sistema de referencias que tomaremos para construirlo. Existen 4 tipos de “sistemas de referencia/coordenadas”, que se muestran gráficamente a continuación.

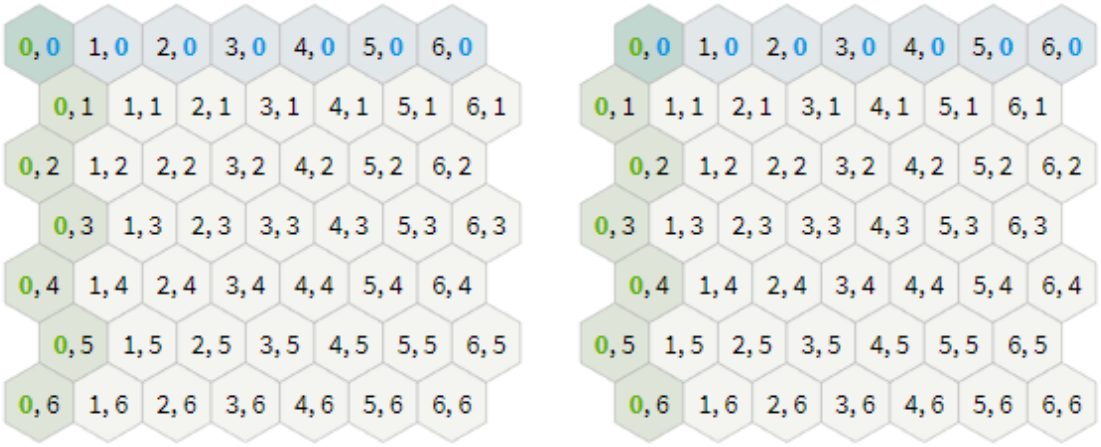


Figura 46. Offset coordinates, De izquierda a derecha, ‘Odd-r’ y ‘even-r’.

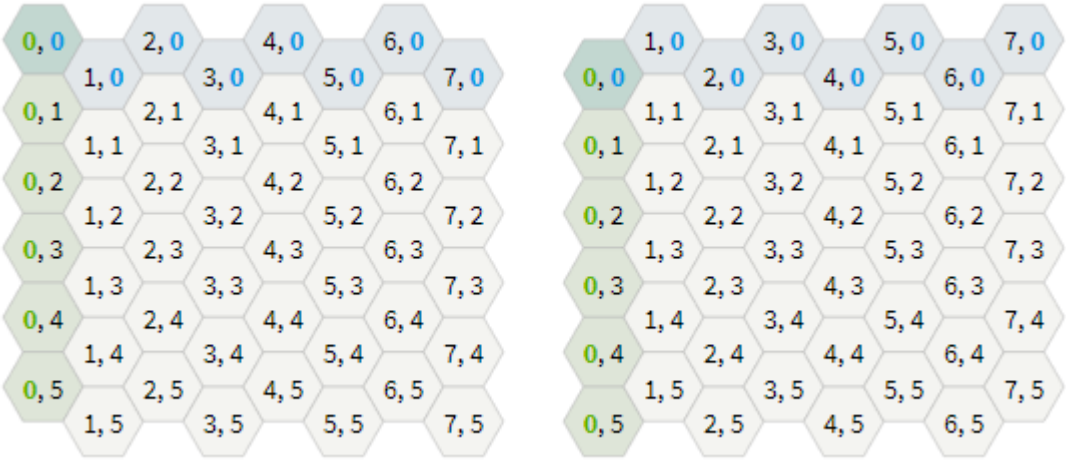


Figura 47. Offset coordinates, De izquierda a derecha, ‘Odd-q’ y ‘even-q’.

5.2.2.1.2. GRID

El tablero es un array bidimensional de casillas (MyTile). Su implementación es bastante extensa, por lo que el contenido técnico se encuentra en el anexo de este mismo documento.

Los parámetros parametrizables de esta clase incluyen las dimensiones del mismo. En el juego, este espacio viene limitado por las decoraciones del escenario, pero si quisiéramos adaptar el escenario, tendríamos que crear un tablero mayor. Al modificar estas características lo que podemos modificar es el número de torres o portales que aparecen a lo largo y ancho del escenario.

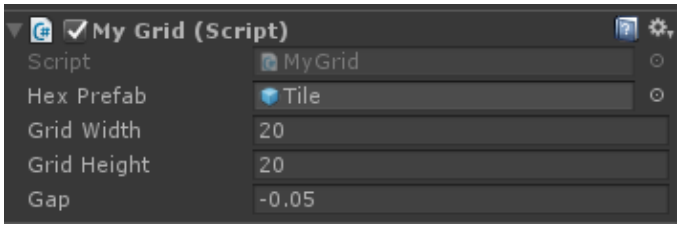


Figura 48. Parámetros del tablero.

Contiene el prefab de la casilla que tomamos como referencia para obtener sus dimensiones. Como parámetros, el alto y ancho de la matriz, así como de un 'gap' u offset que separa las casillas entre sí. En el caso predeterminado, la distancia entre casillas es prácticamente 0, lo que se traduce en que las casillas están prácticamente pegadas entre sí. Como se puede observar en la imagen, las casillas son sencillamente un aspecto visual que se decidió ocultar por razones estéticas.

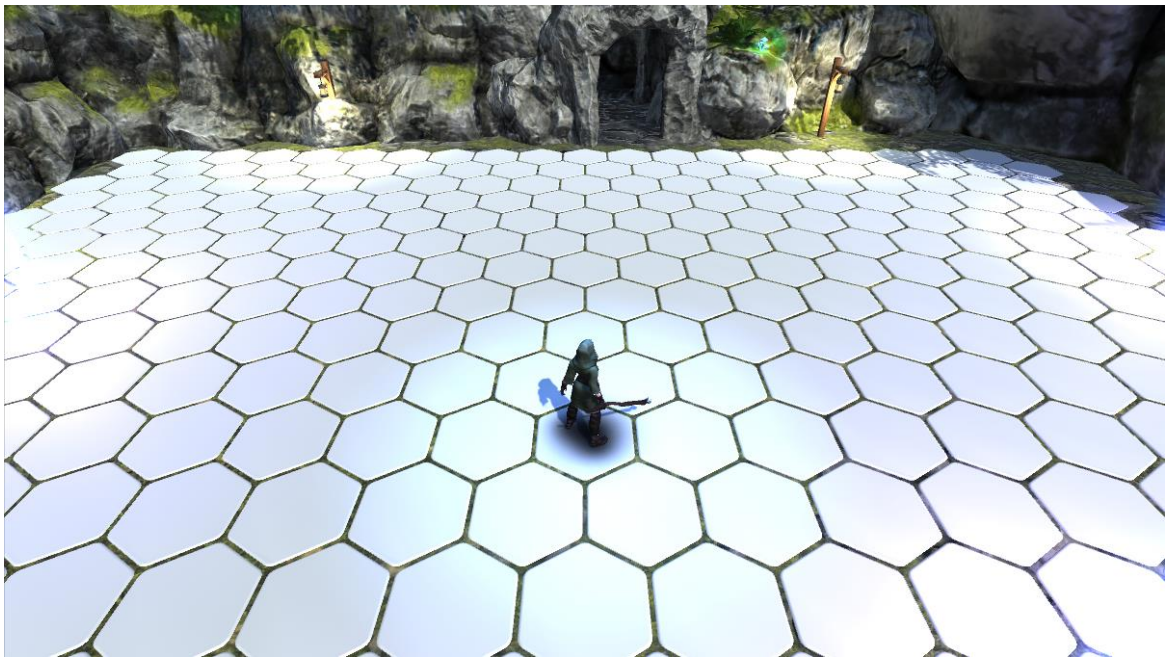


Figura 49. Escenario con las casillas visibles.

5.2.3. ELEMENTOS DE JUEGO

5.2.3.1. ESTRUCTURAS DEFENSIVAS

Como se detalló en el GDD, contamos con torreones que nos ayudarán a combatir a los enemigos.

Figura 50. Captura de las estructuras defensivas in-game.

Estos torreones comparten la clase Tower, donde se detallan las características de cada una de ellas. Radio de ataque, velocidad de ataque, velocidad de proyectil, daño y prefab de la bala son totalmente parametrizables desde esta clase. Además, dispone de un desplegable para seleccionar su tipo.

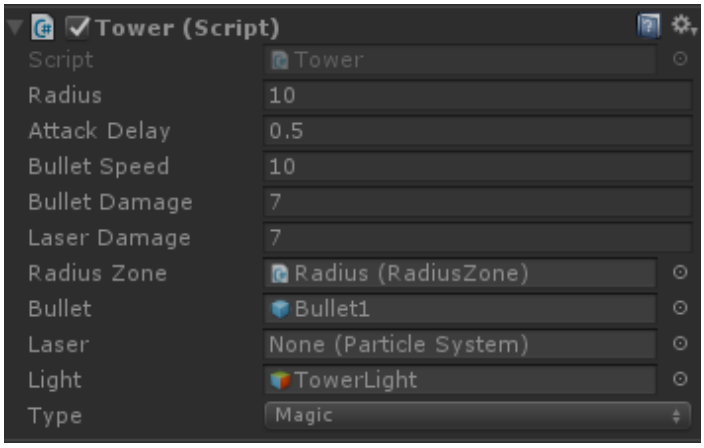


Figura 51. Parámetros de la clase Tower.

Todos los torreones cuentan con el mismo modelo, y se diferencian en la esfera que hay en la parte superior. Los prefabs de los torreones se controlan desde TowerManager, otro objeto Singleton de la escena que contiene los prefabs correspondientes de las torres.

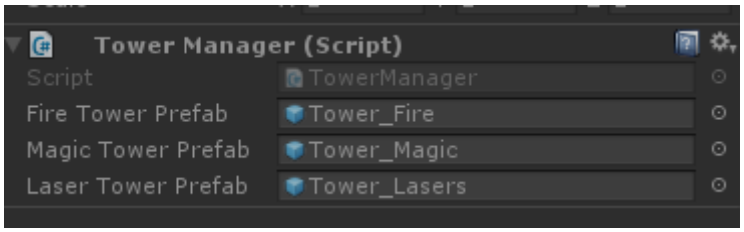


Figura 52. Parámetros de la clase TowerManager.

Además, cada torre tiene añadido en su objeto el efecto de sonido que sonará en el disparo.

5.2.3.1.1. TORRE MÁGICA

Cuenta con una esfera iluminada de color azul como objeto superior. Tiene como efecto de sonido un disparo, como proyectil una esfera luminosa y es de tipo mágica.

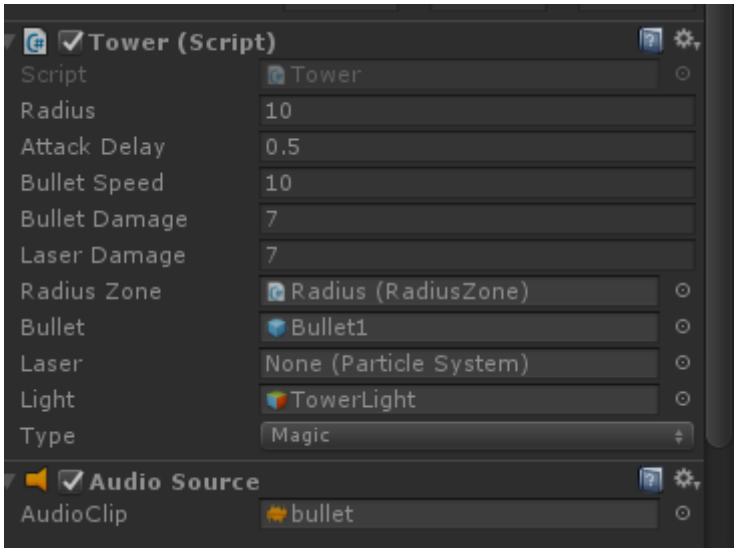


Figura 53. Características de la torre mágica.

5.2.3.1.2. TORRE LÁSER

Tiene un delay mucho menor, una velocidad instantánea y el daño aumentado. Es el único torreón con un sistema de partículas instantáneo asignado, que corresponde al efecto del láser.

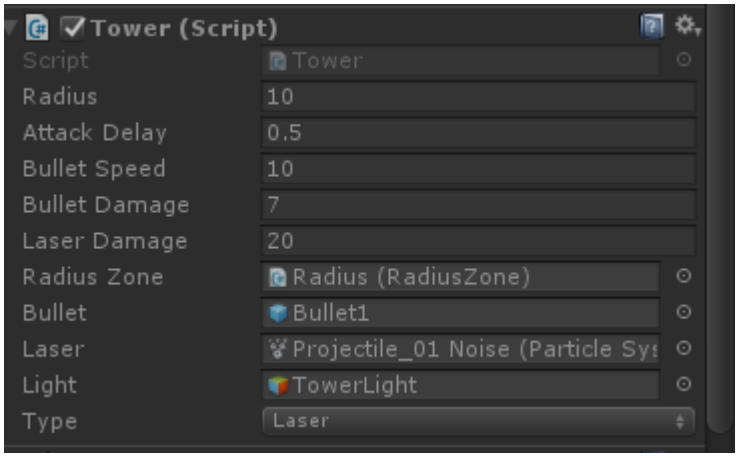


Figura 54. Características de la torre láser.

5.2.3.1.3. TORRE ÍGNEA

Muy parecida a la torre mágica en cuanto a sus parámetros.

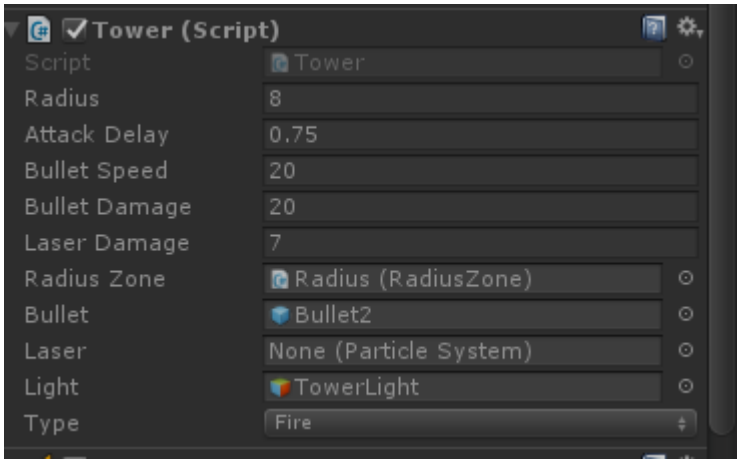


Figura 55. Características de la torre ígnea.

5.2.3.2. SPAWN ENEMIGO

Para gestionar la aparición de los enemigos se utiliza la clase Spawn. Esta clase recibe los posibles enemigos a instanciar y los creará aleatoriamente cada cierto tiempo, también parametrizable. Se pueden añadir los prefabs de forma dinámica en el inspector.

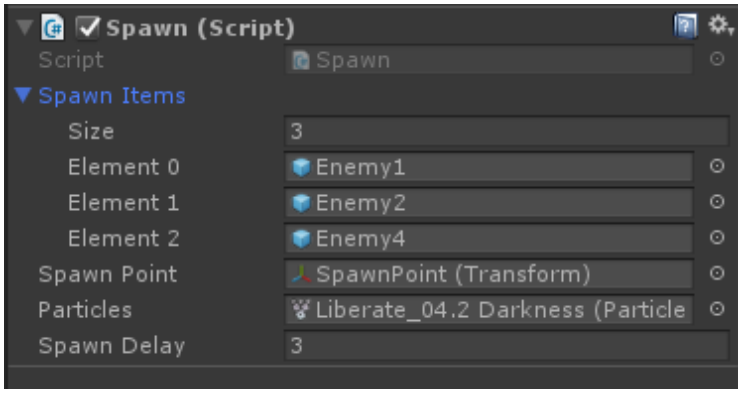


Figura 56. Parámetros del spawn.

De forma predeterminada, el enemigo spawneará en la posición del objeto que contenga el script, pero podrá asignarse un transform externo donde aparecerá, seguido de una animación de partículas que también se establece en el componente.

El tiempo entre aparición de enemigos corresponde al campo Spawn Delay. Independientemente de este valor, los enemigos no aparecerán si es de noche.

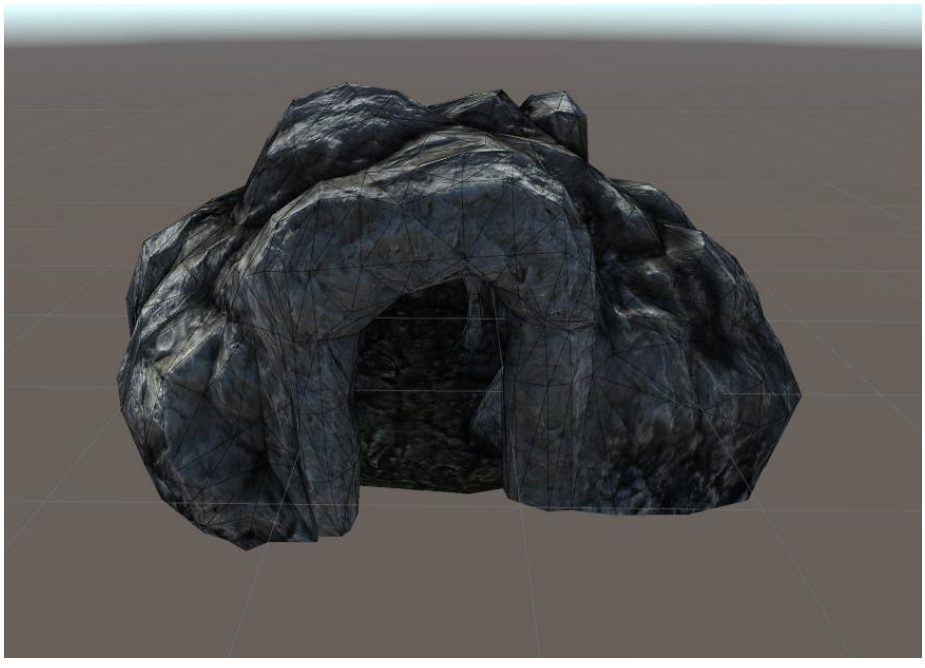


Figura 57. Asset de la cueva enemiga.

5.2.3.3. PORTALES

En la creación aleatoria de portales, estos se asignan de forma automática a la casilla correspondiente, al igual que las torres. Los portales, pues, no contienen parámetros. Todos los valores relacionados con los portales están definidos en LevelManager, que contiene el tiempo de día, noche, portales a aparecer y portales a proteger.

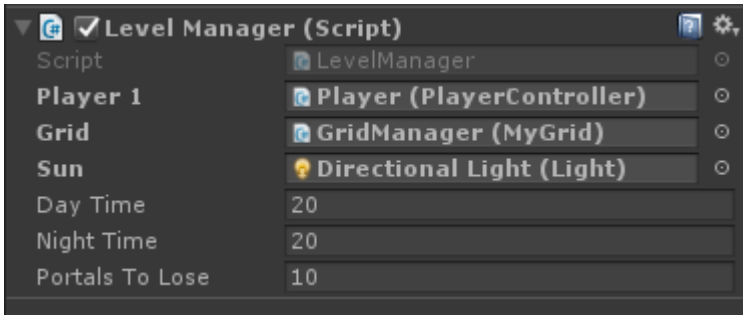


Figura 58. Parámetros de LevelManager.

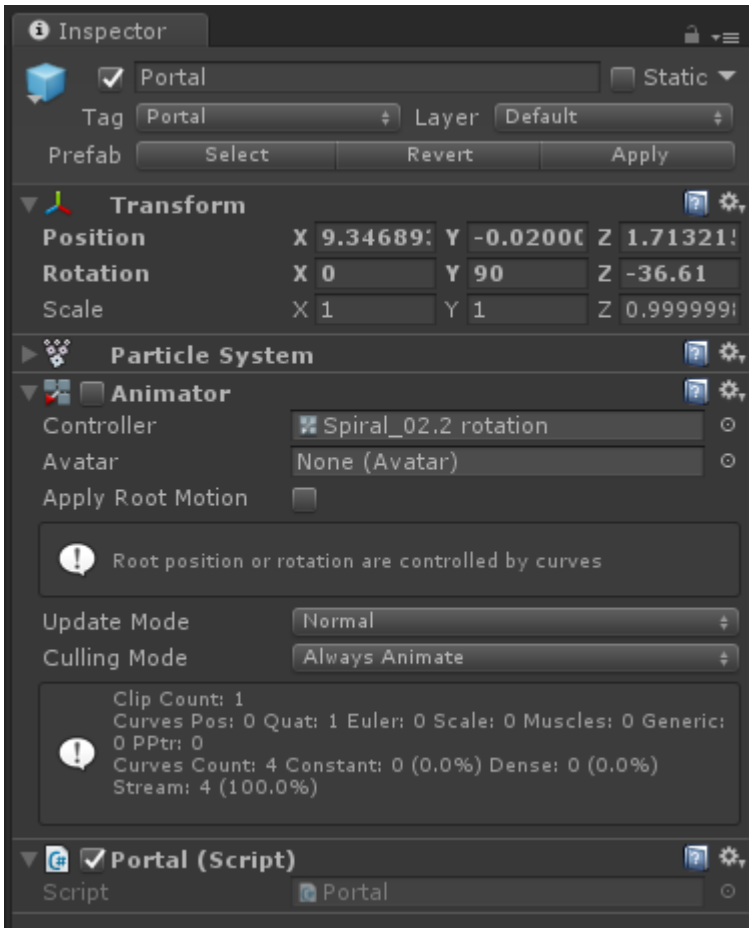


Figura 59. Parámetros del componente Portal.

5.2.4. PERSONAJE PRINCIPAL

El personaje principal es controlado por medio del controlador PlayerController. Este componente contiene los parámetros relacionados con el movimiento, los ataques y los recursos generados (maná) del jugador.

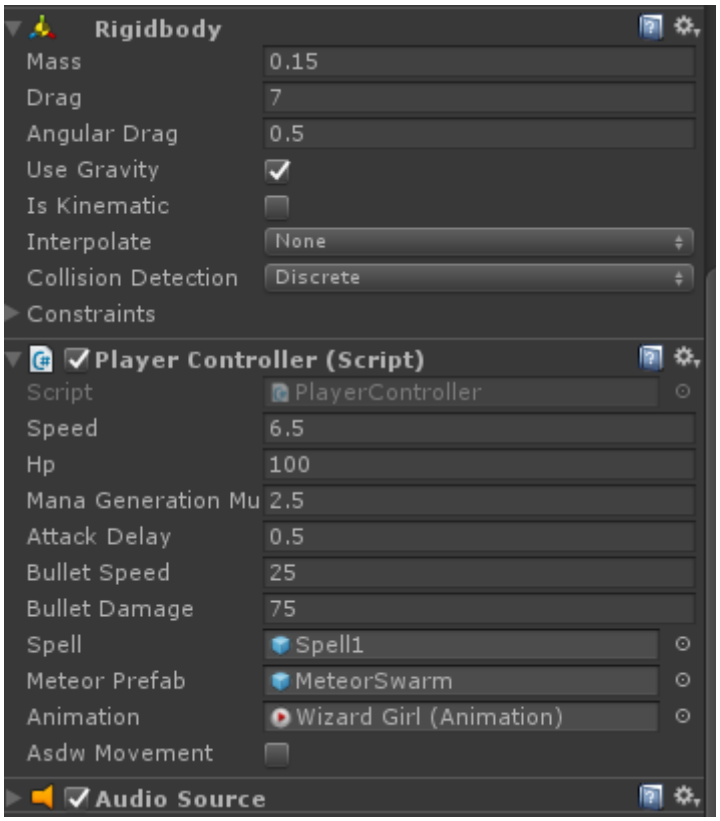


Figura 60. Componente PlayerController

5.2.4.1. MOVIMIENTO Y CONTROL

Como se puede apreciar en la figura 59, el movimiento del personaje se realiza con el componente Rigidbody de Unity, aplicando una fuerza en la dirección del movimiento. Los parámetros de la fuerza a aplicar, que se traduce en velocidad del personaje, están serializados en el componente PlayerController, y se encarga de asignar estos valores en el rigidbody, por lo que no tenemos que preocuparnos de cambiar los valores en los 2 para balancear el movimiento.

Adicionalmente, se ha añadido un movimiento que no se basa en point-to-move, sino el ya conocido sistema cruceta en teclado o ASWD. Este movimiento se añadió como testeo y no está del todo adaptado, ya que al disponer de la posibilidad de mirar en una dirección pero moverse en la contraria, habría que adaptar la animación del personaje para que caminara hacia atrás.

En cuanto a las animaciones, al contrario que con los enemigos, el personaje principal utiliza el componente Animation. Este componente tiene asignadas todas las animaciones individuales, que son llamadas desde PlayerController para reproducirlas.

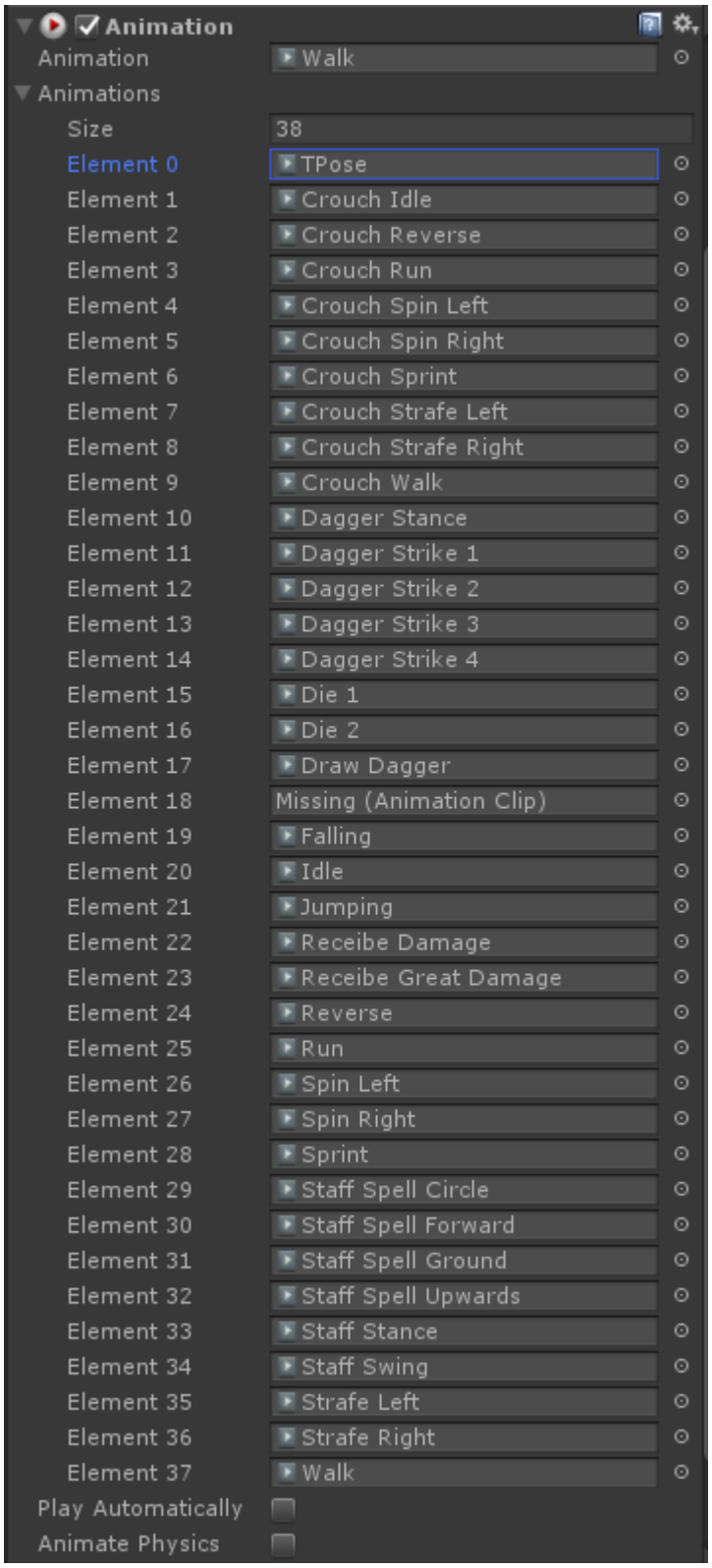


Figura 61. Animaciones del personaje principal.

5.2.4.2. ATAQUE PRINCIPAL

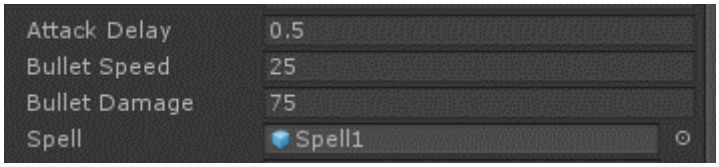


Figura 62. Parámetros del ataque principal.

El ataque principal se basa en un proyectil helado, que se lanza en la dirección del ratón tras una animación y que desaparece al colisionar con un enemigo o cuando sale de los límites del escenario.

Los parámetros relacionados con este ataque son AttackDelay, BulletSpeed y BulletDamage. También es importante el prefab asignado en el campo Spell, ya que es el objeto que se lanza.

Figura 63. Proyectil helado.

5.2.4.3. ATAQUE MÁGICO

El ataque mágico, o ataque final, consiste en una lluvia de meteoritos desde el cielo. Estos meteoritos detectan la colisión con el suelo o con los enemigos, y de ser estos últimos, causan daño de área. En caso de colisionar con el jugador, sencillamente le desplazará en la dirección del impacto.

Este ataque se basa en los parámetros asignados en 2 componentes diferentes, MeteorSwarm, que contiene los valores relacionados con la instanciación de meteoritos, y la clase Meteor, que contiene el daño por meteorito a enemigo. Este valor se ha asignado muy alto para acabar automáticamente con los enemigos.

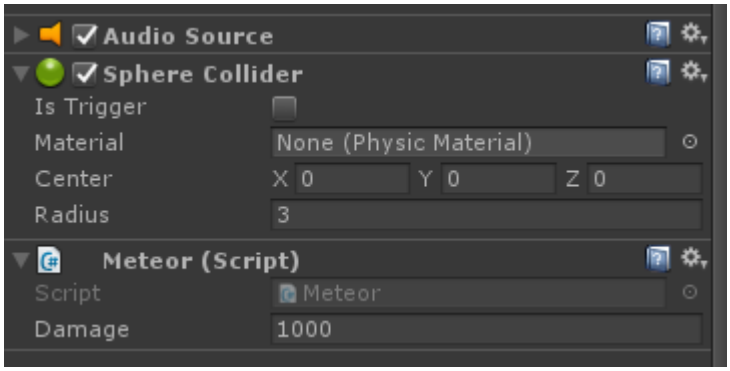


Figura 64. Componente Meteor

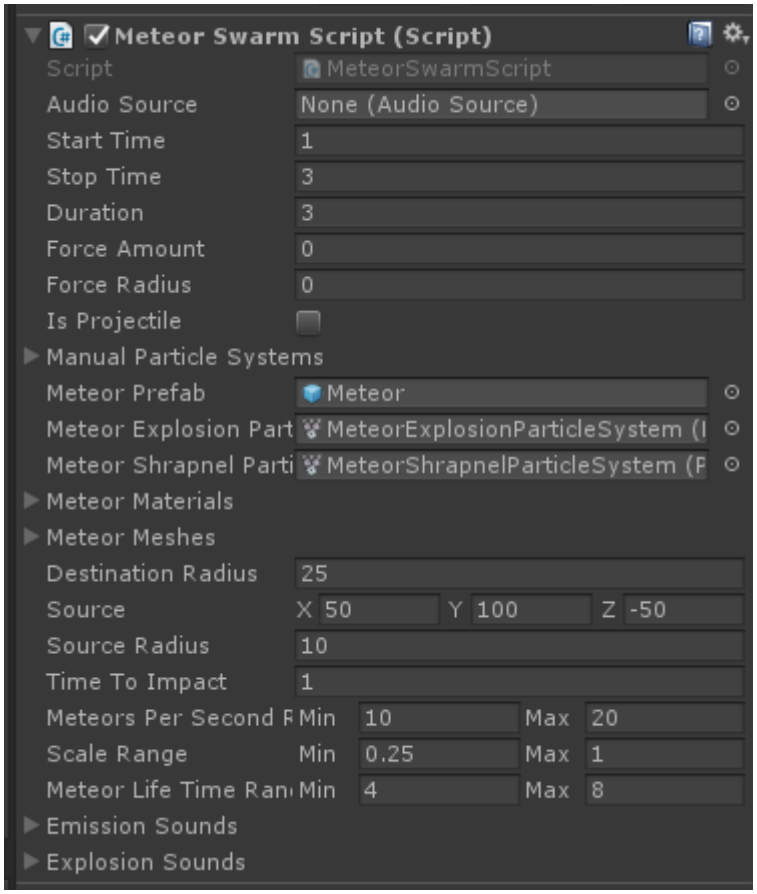


Figura 65. Componente MeteorSwarm

5.2.5. CICLO DE JUEGO

5.2.5.1. *INTERVALO DÍA/NOCHE*

El intervalo de día-noche se establece en el componente LevelManager, de tipo Singleton. Los 2 parámetros que afectan al ciclo de juego son DayTime y NightTime, afectando a tiempo del día y de la noche respectivamente.

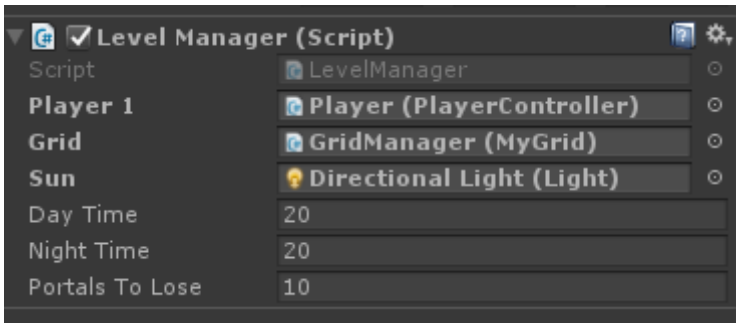


Figura 66. Controlador LevelManager.

En este script podemos ver cómo está definido el ciclo de juego.

```
1.  if (_timer <= 0 f) { // Score
2.      if (_day) _dayScore++;
3.      else _nightScore++;
4.      if (!_day) {  // Stop enemy spawning and clean enemies and portals
5.          SpawnEnemies(false);
6.          KillEnemies();
7.          KillPortals();
8.          UIController.Instance.SetWaves(_nightScore);
9.      }
10.     _day = !_day;
11.     _timer = _day ? _dayTime : _nightTime;
12.     if (_day) {
13.         StartCoroutine(SpawnPortals());  // Create new portals
14.         UIController.Instance.SetButtons(true);  // Enable tower building buttons
15.     }
16.     if (!_day) {
17.         UIController.Instance.SetButtons(false);  // Disable tower building buttons
18.         SpawnEnemies(true);  // Enable enemy spawns
19.     }
20. }
```

Figura 67. Ciclo de juego

Además, el ciclo de juego predeterminado incluye que se comience en pleno día, permitiendo así construir defensas antes de la primera oleada.

LevelManager también es el encargado de controlar la iluminación de la escena dependiendo del tiempo restante para anochecer o amanecer, y estableciendo una luminosidad proporcional al tiempo restante.

5.2.6. ENEMIGOS

El controlador de cada enemigo es la clase Enemy. Además, tienen un componente NavMesh, propio de Unity, que será el encargado de calcular las rutas hacia los portales, como se detallará en el apartado de IA.

5.2.6.1. CLASES DE ENEMIGOS

Siguiendo las especificaciones del GDD, contamos con 3 enemigos: gólem, ogro y esqueleto. Todos siguen la misma estructura, partiendo de los valores individuales que se les establezca en el componente Enemy.

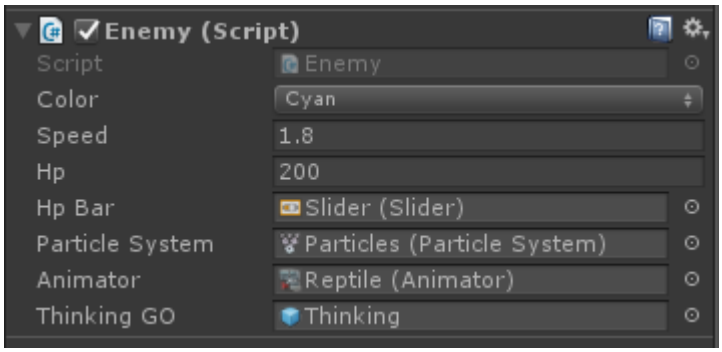


Figura 68. Componente Enemy.

Este script controla las variables de los enemigos: velocidad, vida y color del ambiente. El color es un aspecto meramente visual, y se refiere al color de las partículas (otro parámetro del mismo componente) que rodean al modelo.

Figura 69. Color del enemigo.

En cuanto a las animaciones, los enemigos cuentan con un componente animator, que, a diferencia del personaje principal, permite crear transiciones entre los diferentes estados. Unity ofreces opciones de loop, transición y velocidad individual de cada una de las animaciones, lo que aporta un gran acabado para suavizar los cambios bruscos entre una animación y la siguiente. En la siguiente figura se puede observar el mapa de transiciones de los enemigos.

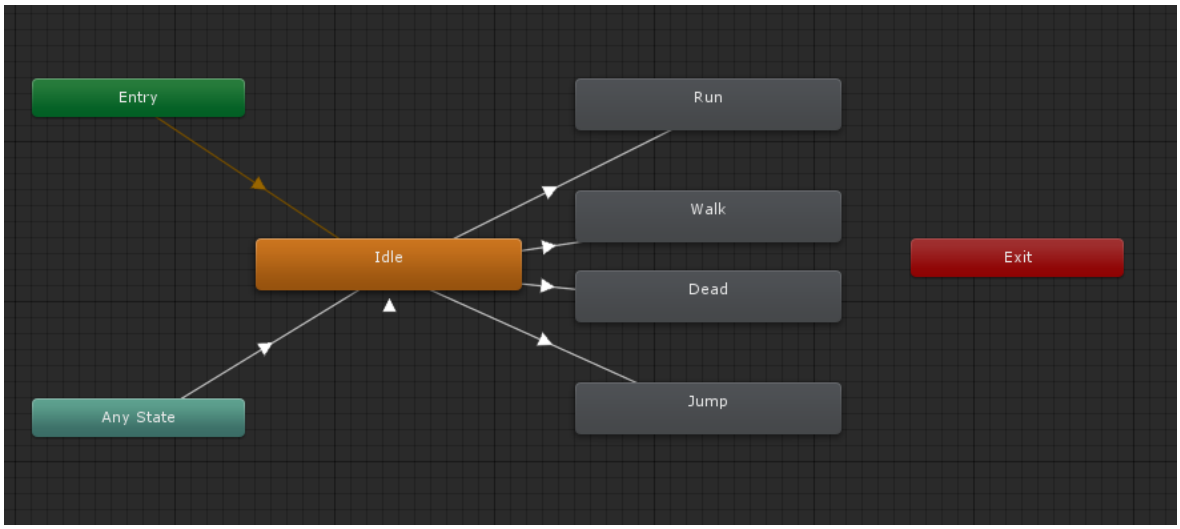


Figura 70. Diagrama de las animaciones enemigas.

Cada enemigo cuenta con diferentes animaciones asociadas a su modelo, pero el árbol de transiciones es el mismo para ambos.

Otro aspecto que puede echarse en falta es un campo de tipo “Type”, como el existente en la clase Tower. Esto carece de sentido, pues en el sistema de juego es irrelevante el tipo de enemigo para cualquier aspecto. No es necesario esta información para perder o ganar la partida ni para afectar de distinta manera dependiendo de la torre. Todos los enemigos son tratados por igual en portales, hechizos y proyectiles, por lo que toda la información individual está contenida en los parámetros del prefab correspondiente.

Los enemigos también comparten un parámetro “Thinking” que veremos en el siguiente apartado.

5.2.6.2. IA

La IA enemiga es manejada y controlada mediante un NavMesh. Es una clase de unity responsable de calcular un camino viable y óptimo para unos agentes, que en nuestro caso serán los enemigos. El navmesh es una superfície parametrizable desde el inspector de Unity que nos permite establecer valores como anchura máxima de los agentes, separación con las paredes o límites de la superficie y

otros parámetros relacionados con el mismo. En el escenario, esta superfície se corresponde con el suelo del escenario, y los puntos objetivos serán las posiciones de los portales. Además, esta superfície se irá editando paulatinamente a lo largo del juego a medida que se vayan construyendo torres, que actuarán como obstáculos.

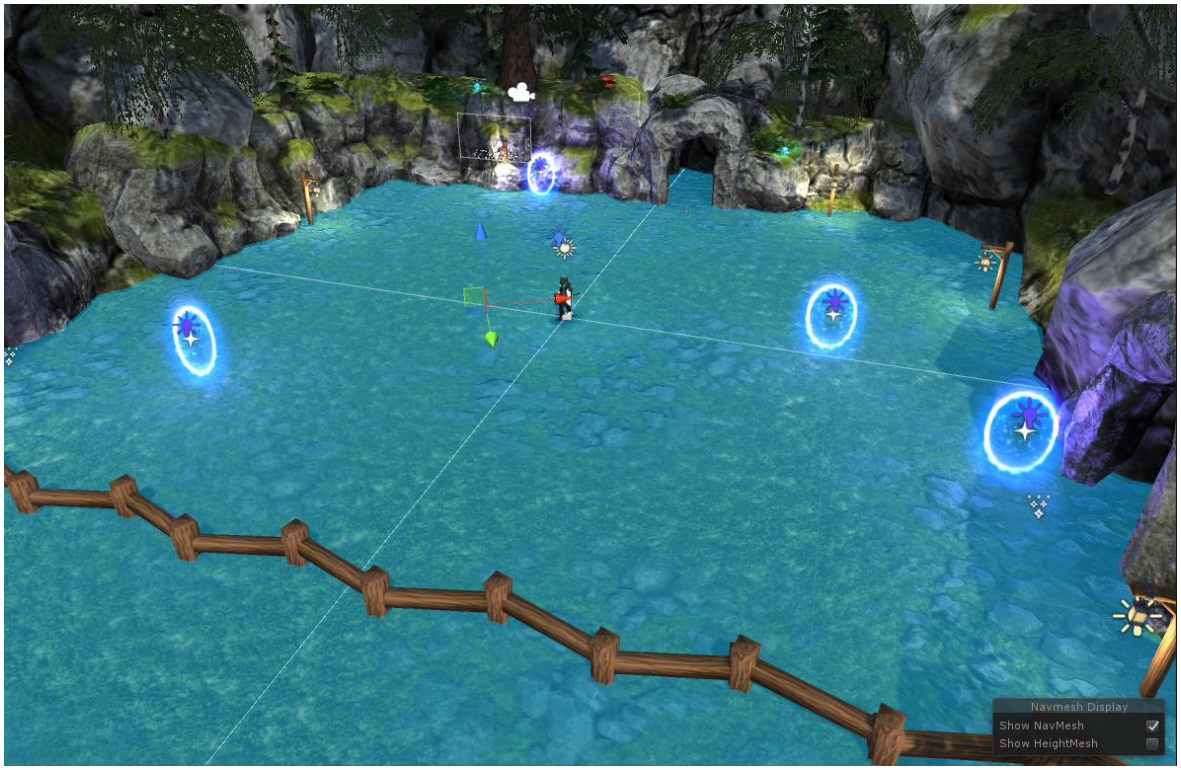


Figura 71. Superfície del NavMesh en el escenario.

Los enemigos cuentan con el componente NavMeshAgent, que calcula automáticamente el camino deseado, evitando los obstáculos.

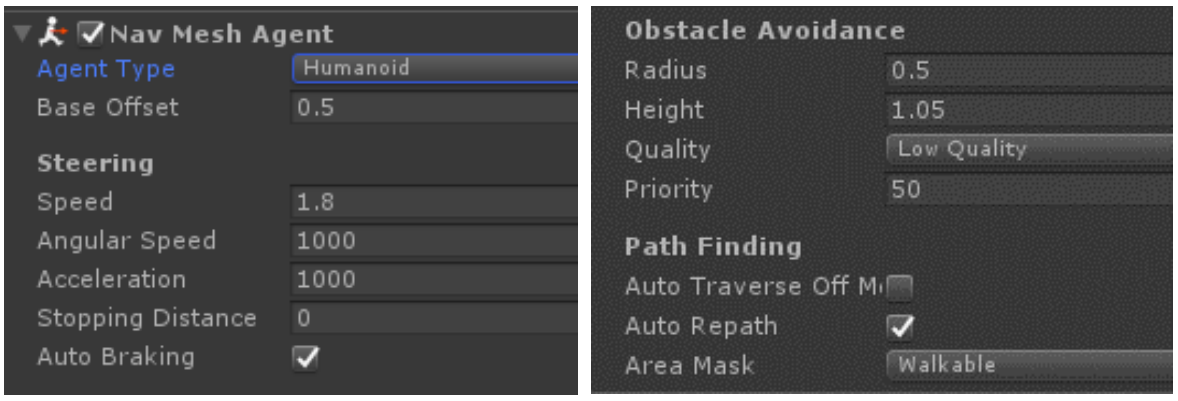


Figura 72. Componente NavMeshAgent

Generalmente, el NavMesh debe bakearse antes de ejecutar la escena para tener cálculos previos. Sin embargo, en las últimas versiones de Unity se han incluido mejoras y ejemplos que permiten recalcular en tiempo real los nuevos caminos que pueden producirse cuando se introduce un cambio en la superfície, como una modificación del escenario o la inclusión de obstáculos, que en nuestro caso serán las estructuras defensivas. Estos componentes son los siguientes:

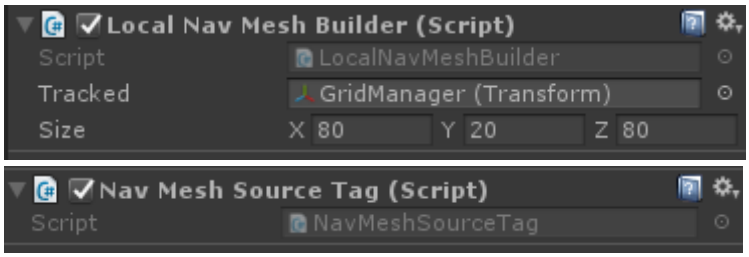


Figura 73. Componentes para reconstruir el NavMesh

Son componentes que actualmente están en fase de pruebas en la versión de Unity empleada (5.6), pero que funcionan bastante bien. Su único punto desfavorable es el rendimiento cuando se sobresaturan los obstáculos y el número de agentes, pero dado que están limitados en el nivel, utilizar este método supone más ventajas que inconvenientes.

Figura 74. NavMesh recalculado tras la construcción de obstáculos.

5.2.7. UI

5.2.7.1. HUD

El HUD, o interfaz dentro de la partida, es controlado por otro Singleton llamado UIController. Es el encargado de actualizar el texto mostrado en pantalla y actualizar los valores. Gran parte de este controlador hace uso de LocalizationManager para traducir al idioma actual los textos a mostrar.

Los campos manejados por el controlador son las puntuaciones y contador del menú superior y el maná disponible del panel lateral. También es el responsable de manejar los inputs en los botones laterales, activarlos y desactivarlos, así como el botón del hechizo final, la lluvia de meteoritos.

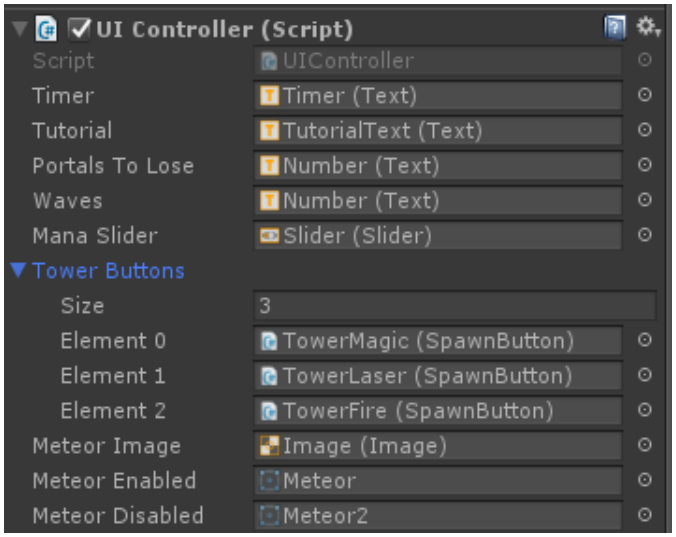


Figura 75. UIController.

Para la realización de los assets me he basado en un diseño previo, que se puede ver a continuación.

Figura 76. Diseño básico de la interfaz.

Estos assets se han modificado, adaptando los colores y huecos disponibles correspondiente a las torres y añadiendo texturas. El resultado final son los assets incluidos.

Figura 77. Assets finales para la interfaz.

En cuanto a las imágenes representativas de las estructuras y el ataque de meteoritos son capturas in-game. También se ha añadido un tachado para cuando los botones no están disponibles.

Figura 78. Imágenes de los botones laterales.

5.2.7.2. MENÚS

La pantalla de menú principal es muy sencilla. Cuenta con un controlador, MenuController, que es el encargado de gestionar los callbacks de los botones y ocultar y mostrar los paneles correspondientes, así como de llamar a LocalizationManager para cambiar de idioma en el menú de opciones.

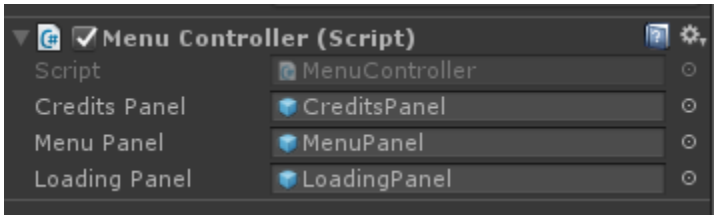


Figura 79. Componente Menu Controller.

El estilo del botón se ha modificado a partir de un recurso gratuito, dejando un tono grisáceo cuando es pulsado.

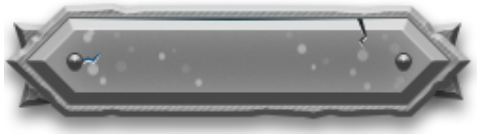


Figura 80. Botón genérico del menú.

5.2.8. CÁMARA

La cámara, controlada desde CameraController, sigue al personaje principal, y está preparada para mantener en pantalla a 2 elementos, desplazándose con un suavizado. Además, su posición, ángulo, velocidad y zoom es configurable desde el inspector.

Figura 81. Componente CameraController.

5.2.9. AUDIO

El sistema de audio funciona basándose en AudioManager, otra clase Singleton, que es el encargado de gestionar los efectos de música ambiente. Como parámetros recibe dos audios, la música del menú y del juego.

Los efectos de sonidos son reproducidos por cada uno de los objetos que los emiten, por lo que no dependen del AudioManager. Esto es así para evitar clipping reproduciendo el mismo SFX al mismo tiempo desde varias fuentes.

Figura 82. Parámetros de AudioManager

AudioManager cuenta con un componente AudioSource que será el encargado de reproducir la música de fondo actual.

Los demás efectos de sonidos no están serializados pues, sino que se acceden a ellos desde la carga de recursos.

5.2.10. SISTEMA DE IDIOMAS

El sistema de idiomas se basa en un parseador de archivos tsv. En la escena contamos con un gameobject al que añadimos el script LocalizationManager.

LocalizationManager es una clase Singleton a la que podremos llamar desde cualquier parte de otro script para recibir el texto localizado. Desde el inspector de Unity, este gameobject se inicializa con los archivos que contienen los idiomas del juego.

Estos archivos de idiomas son archivos .txt que contienen todas las claves y valores del juego. A la hora de leerlas, el manager busca la key solicitada en el idioma establecido y devuelve su valor asociado, si existe. Los idiomas se pueden añadir de forma dinámica desde el inspector.

Figura 83. Parámetros de LocalizationManager.

El formato de estos archivos consta de una línea por cada par de clave/valor, separadas por un espacio y el símbolo igual entre ellos. Soporta codificación UTF-8, por lo que permite guardar acentos y caracteres especiales. Para realizar la prueba, se añadió el idioma japonés.

```
1. _BACK = Volver
2. _LOADING = Cargando...
3. _QUIT = Salir
4. _START_GAME = Jugar
5. _CREDITS = Créditos
```

Figura 84. Ejemplo de archivo de localización en español.

Así pues, para recibir el texto localizado, llamaremos al método GetKey(string key), con la Key del texto a localizar.

Además, para no tener que asociar todos los textos desde código, contamos también con otra clase llamada Localizer. Este script, asociado a cualquier gameObject que contenga un componente de tipo texto establecerá el texto por defecto que se le indique en el campo público key en el idioma seleccionado en el momento de la llamada.

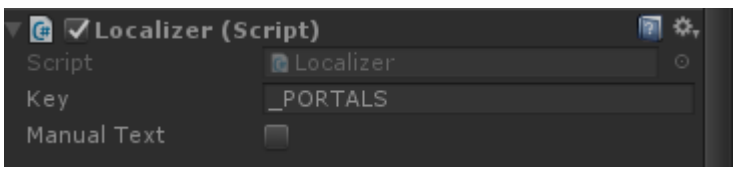


Figura 85. Componente Lozalizer

Por ejemplo, si creamos un panel con un texto que queremos que contenga de forma fija la palabra “Salir del juego” en el idioma correspondiente, añadimos el componente Localize al gameObject y colocaremos la clave “_EXIT_GAME”. Esta clave deberá estar en todos los archivos de localización asociados en el LocalizationManager, y se establecerá por defecto a su Start o si cambiamos el idioma del juego.

6. CONCLUSIONES

6.1. REPASO DE OBJETIVOS

- Implementar todo el proyecto utilizando herramientas gratuítas y sin ningún tipo de presupuesto.

Este objetivo se ha cumplido. Los assets visuales (modelos y animaciones) se han obtenido de la asset store. Son recursos gratuitos que se han modificado para adecuarlos al juego, al igual que los archivos de sonido

- Realizar el documento de diseño (GDD) del mismo.

Realizado (Apartado 5.1 del documento).

- Planificar correctamente el desarrollo del proyecto.
- Aprender a organizar y gestionar el proyecto empleando metodologías de trabajo.
- Implementar las mecánicas del videojuego.
- Implementar un código correcto, funcional, limpio y ampliable.

Una de las características del proyecto es que se ha preparado para que cambiar enemigos, torres, recursos y añadir o quitar contenido sea trivial. Un buen ejemplo lo encontramos en cualquier Manager.

6.2. TIEMPO DEDICADO

Para la realización de este proyecto he dedicado por igual tiempo en diseño e implementación, y la mayoría del mismo se ha dedicado a prototipar y testear funcionalidades, efectos y recursos.

Incluso ha habido una inversión de tiempo dedicado a mejorar y optimizar el producto para tratar de adaptarlos a plataformas móviles o tablets, pero por cambios de jugabilidad y limitación de recursos se optó por descartar esta idea.

En cuanto al tiempo de desarrollo, ha sido a lo largo de 7 meses con una media de 6~8h por semana. En total se calculan que se habrán invertido cerca de las 220h entre diseño, prototipado, implementación y documentación.

6.3. Propuestas de mejora

9. Balanceo de juego: El juego comienza siendo complicado al principio. Llegan muchos enemigos seguidos y tenemos poco tiempo de reacción a menos que ya hayamos jugado anteriormente. Balancear la curva de dificultad no es tarea sencilla, pero está dentro de las mejoras deseables.

10. Modo multijugador: En un principio se barajó la posibilidad de hacer un modo cooperativo local, pero la falta de inputs de entrada nos restringe esta posibilidad, ya que la interfaz está adaptada a jugar con teclado y ratón, y no con mando, que sería una posibilidad.

    Otra opción sería conseguir un modo multijugador en red basado en salas, pero requiere un desarrollo que queda fuera del alcance de este proyecto.

11. Crear niveles visuales: El lanzador de Unity nos permite seleccionar el nivel gráfico (Muy bajo, bajo, medio, alto, fantástico), pero no es posible desde dentro del juego.

12. Más contenido: Torres, enemigos, ataques, sonidos. Sobretodo torres y enemigos, ya que el sistema está preparado para insertar este contenido de forma muy sencilla, con sólo añadir el prefab de los enemigos o torres en el inspector, en los campos serializados. De la misma manera, se podrían añadir idiomas, ya que se definen en los archivos de localización y añadirlos al manager es algo trivial.

13. Pantalla de puntuaciones/ranking: Esto es algo que ha quedado en el tintero, ya que la idea original era crear un botón social para compartir nuestra puntuación en Facebook o Twitter, pero estaba más orientadas a plataformas móviles, no de sobremesa. Como finalmente se descartó la plataforma móvil, esta mecánica perdió atractivo.

6.4. Reflexión general

En general, estoy satisfecho con el trabajo realizado, ya que he aprendido diversas funcionalidades de Unity, tanto de desarrollo como de implementación, así como a trabajar con aspectos de arte a los que no estoy acostumbrado. Además, gestionar un proyecto que desemboca en un resultado tangible y sobretodo, jugable, resulta reconfortante.

Con todos los conocimientos aprendidos, es fácil detectar aspectos que podrían haberse hecho de otra manera y que podrían mejorarse. Personalmente, que ocurra esto lo considero algo positivo, ya que es una muestra de que se ha aprendido algo y que en ocasiones futuras se hará mejor.

Estoy bastante contento con el resultado visual del juego, ya que ha quedado bastante acorde a la idea general que se diseñó, pese a no tener bocetos de ello. Sin embargo, hacer uso exhaustivo de los sistemas de partículas, normales, vegetación y animaciones hacen mella en el rendimiento de la aplicación. La iluminación, uno de los aspectos más importantes del juego, se cobra parte del performance, gran parte por uso de sombras. Son detalles técnicos que, de haber tenido constancia de los detalles técnicos del motor, se habrían diseñado de otra manera para no cargar tanto la escena.

En resumen, me siento satisfecho con el trabajo realizado y el resultado final del proyecto, además de tener la motivación para seguir mejorándolo y añadiendo contenido.

7. BIBLIOGRAFÍA Y REFERENCIAS

Por orden de aparición:

[1] Definición de videojuego

https://es.wikipedia.org/wiki/Videojuego

[2] Historia de los tower defense

http://ezinearticles.com/?History-of-Tower-Defense-Games&id=7241055

[3] Motores gráficos

http://www.aragon.es/estaticos/GobiernoAragon/Departamentos/InvestigacionInnovacionUniversidad/Areas/Sociedad_Informacion/Documentos/Estado%20del%20arte%20GameEngines%20y%20su%20impacto%20en%20la%20industria.pdf

[4] Logo de Unity

https://upload.wikimedia.org/wikipedia/commons/5/55/Unity3D_Logo.jpg

[5] Logo de Unreal Engine

https://upload.wikimedia.org/wikipedia/commons/e/ee/Unreal_Engine_logo_and_wordmark.png

[6] Ori and the blind forest

http://es.ign.com/ori-and-the-blind-forest-xbox-one/91739/review/ori-and-the-blind-forest-analisis-para-xbox-one-360-y-pc

[7] Referencia de Bosque encantado

https://www.deviantart.com/art/Forest-Wisp-452203788

[8] Método Kanban

http://www.javiergarzas.com/2011/11/kanban.html

[9] Método Scrum

https://proyectosagiles.org/que-es-scrum/

[10] Logo PEGI-12

https://upload.wikimedia.org/wikipedia/commons/thumb/4/44/PEGI_12.svg/426px-PEGI_12.svg.png

[11] Creación de un grid hexagonal

http://catlikecoding.com/unity/tutorials/hex-map/part-1/

https://tbswithunity3d.wordpress.com/2012/02/23/hexagonal-grid-path-finding-using-a-algorithm/

http://www.redblobgames.com/grids/hexagons/

http://gamelogic.co.za/grids/documentation-contents/quick-start-tutorial/gamelogics-hex-grids-for-unity-and-amit-patels-guide-for-hex-grids/

[12] Unity answers

http://answers.unity3d.com/index.html

8. ANEXO

- Enlace al repositorio

https://github.com/Venisir/Twilight-Whispers

- Gameplay

https://www.youtube.com/watch?v=VTAFm5FXwyA

- Avance visual del proyecto

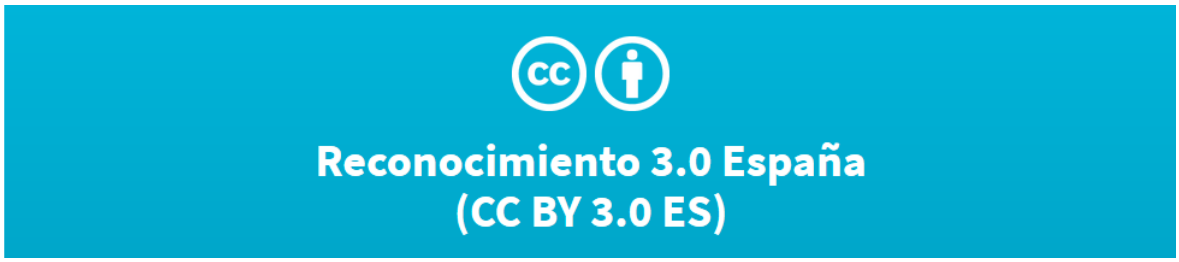
https://drive.google.com/file/d/0BzQxa8JqnAzWTHJOYzBZMFZxT1U/view?usp=sharing

8.1. RECURSOS

Recursos empleados para añadir contenido al proyecto. Todos los recursos utilizados están bajo la licencia Creative Commons 3.0. Las condiciones de esta licencia pueden comprobarse en su web:

Attribution 3.0 Unported (CC BY 3.0)

https://creativecommons.org/licenses/by/3.0/



- Fuente de logotipo: Alice and the Wicked Monster

http://www.dafont.com/es/alice-and-the-wicked-monster.font

- Fuente de menús: Diogenes

http://www.dafont.com/es/diogenes-2.font

- Fuente de tutorial e interfaz: DK Moonlight Shadow

http://www.dafont.com/es/moonlight-shadow.font

8.1.1. ARTE

- Textura de torre

https://www.assetstore.unity3d.com/en/#!/content/73949

- Otras texturas

https://www.textures.com/download/soilsand0168/45790?q=seamless

- Gólem

https://www.assetstore.unity3d.com/en/#!/content/48933

- Ogro

https://www.assetstore.unity3d.com/en/#!/content/84219

- Esqueleto

https://www.assetstore.unity3d.com/en/#!/content/35635

- Sistemas de partículas

https://www.assetstore.unity3d.com/en/#!/content/46176

https://www.assetstore.unity3d.com/en/#!/content/36825

https://www.assetstore.unity3d.com/en/#!/content/42866

https://www.assetstore.unity3d.com/en/#!/content/36825

8.1.2. AUDIO

- Audio menú principal

Eternal Terminal Kevin MacLeod (incompetech.com), Licensed under Creative Commons

- Audio principal del juego

Wretched Destroyer Kevin MacLeod (incompetech.com), Licensed under Creative Commons

- Disparo – Torre mágica

https://freesound.org/people/LiamG_SFX/sounds/323086/

- Disparo – Torre láser

https://freesound.org/people/LiamG_SFX/sounds/334241/

- Disparo – Torre ígnea

https://freesound.org/people/LiamG_SFX/sounds/334234/

- Spawn

https://freesound.org/people/Speedenza/sounds/168180/

8.2. CAPTURAS DEL VIDEOJUEGO

Figura 86. Captura In-game del juego.

Figura 87. Captura In-game del juego.

Figura 88. Captura In-game del juego.

Figura 89. Captura In-game del juego.

Figura 90. Captura In-game del juego.

Figura 91. Captura In-game del juego.

Figura 92. Captura In-game del juego.

Figura 93. Captura In-game del juego.